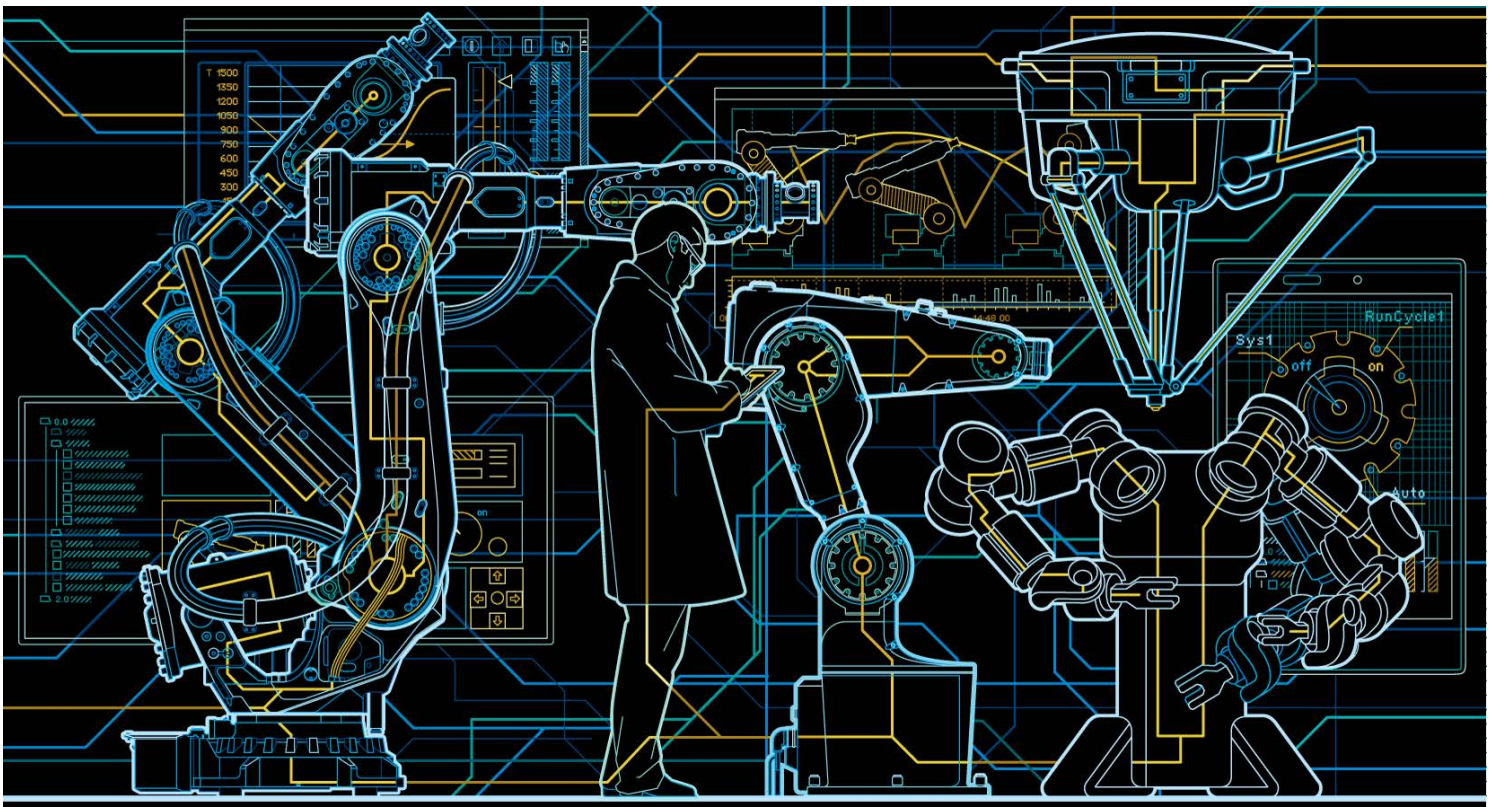


Technická referenční příručka
RAPID - Instrukce, funkce a datové typy

Power and productivity
for a better world™

Trace back information:
Workspace R16-1 version a18
Checked in 2016-06-09
Skribenta version 4.6.318

Technická referenční příručka
RAPID - Instrukce, funkce a datové typy

RobotWare 6.03

ID dokumentu: 3HAC050917-014
Revize: C

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Informace v této příručce mohou být změněny bez předchozího upozornění a nelze je považovat za závazné pro společnost ABB. ABB nepřijímá zodpovědnost za žádné chyby, které se v této příručce mohou vyskytnout.

S výjimkou případů výslovně vyznačených v této příručce nelze z uváděných informací vyvozovat jakýkoli druh záruky společnosti ABB za ztráty, škody na zdraví či majetku, vhodnost pro určitý účel apod.

Společnost ABB v žádném případě neodpovídá za náhodné nebo následné škody vzniklé při používání této příručky a výrobků, které jsou zde popisovány.

Tato příručka ani její části nesmějí být reprodukovány ani kopírovány bez písemného souhlasu společnosti ABB.

Další výtisky této příručky lze získat od společnosti ABB.

Původním jazykem této příručky je angličtina. Všechny ostatní dodávané jazykové verze byly z angličtiny přeloženy.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

ABB AB
Robotics Products
Se-721 68 Västerås
Švédsko

Obsah

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Přehled této příručky

O této příručce

Toto je technická referenční příručka pro programátora RAPID. V této příručce jsou podrobně rozvedeny základní instrukce, funkce a druhy dat RAPID.

Použití

Tuto příručku byste měli během programování sledovat, když budete potřebovat konkrétní informace o instrukcích, funkcích a druzích dat RAPID.

Kdo by si měl přečíst tuto příručku?

Tato příručka je určena pro uživatele s předchozí zkušeností s programováním, např. programátory robotů.

Požadavky

Čtenář by měl mít zkušenosti s programováním a prostudované

- *Návod k použití - Introduction to RAPID*
- *Technická referenční příručka - Přehled RAPID*

Uspořádání kapitol

Příručka je rozčleněna do následujících kapitol:

Kapitola	Obsah
1 Instrukce	Podrobné popisy všech základních instrukcí RAPID včetně příkladů, jak je používat.
2 Funkce	Podrobné popisy všech základních funkcí RAPID včetně příkladů, jak je používat.
3 Druhy dat	Podrobné popisy všech základních druhů dat RAPID včetně příkladů, jak je používat.
4 Příklady typu programování	Všeobecný přehled o psaní programového kódu, který obsahuje různé instrukce/funkce/druhy dat. Kapitola obsahuje také programovací tipy a vysvětlení.

Reference

Reference	ID dokumentu
Technická referenční příručka - Instrukce, funkce a druhy dat RAPID IRC5 s RobotWare 5.	3HAC16581-001
Návod k použití - Introduction to RAPID	*3HAC029364-014*
Technická referenční příručka - Přehled RAPID	*3HAC050947-014*
Technická referenční příručka - RAPID kernel	*3HAC050946-014*
Application manual - Controller software IRC5	*3HAC050798-014*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pokračování

Revize

Revize	Popis
-	Vydáno s verzí RobotWare 6.0. **POZNÁMKA** U IRC5 s RobotWare 5, viz příručku 3HAC16581-001.
A	Vydáno s verzí RobotWare 6.01. • Doplněny byly následující instrukce, funkce a druhy dat: *AliasIOReset - Resetování I/O signálu s alias jménem na str 31*, *TriggJIOs - Společné pohyby robotu s I/O událostmi na str 847* • Informace o 7-osých robotech byla doplněna k druhu dat *confdata - Konfigurační data robotu na str 1473*.
B	Vydáno s verzí RobotWare 6.02. • Doplněny byly následující obvyklé instrukce, funkce a druhy dat: *SaveCfgData - Uložit systémové parametry do souboru na str 581*, *cfgdomain - Konfigurační doména na str 1471*, *TriggDataCopy - Kopírovat obsah do proměnné triggdata na str 810*, *TriggDataReset - Resetovat obsah v proměnné triggdata na str 812*, *TriggDataValid - Zkontrolujte, jestli obsah proměnné triggdata je platný na str 1374* *AInput - Přečte hodnotu analogového vstupního signálu na str 1033*, *DInput - Čte hodnotu digitálního vstupního signálu na str 1134*, *GInput - Číst hodnotu skupinového vstupního signálu na str 1189* • Byly doplněny všechny instrukce, funkce a druhy dat pro RobotWare doplněk*Integrated Vision*. • Varování o mezní vzdálenosti bylo doplněno k *SoftAct - Aktivace měkkého serva na str 685*. • Byly přidány trigonometrické funkce pro datový typ `dnum`: `ACosDnum, ASinDnum, ATanDnum, ATan2Dnum, CosDnum, TanDnum, SinDnum` • Byly doplněny RAPID instrukce pro funkcionalitu EGM Path Correction: *EGMActJoint - Připravit pohyb EGM pro cíl spoje na str 159*, *EGMMoveC - Kruhový EGM pohyb s korekcí dráhy na str 169*, *EGMMoveL - Lineární EGM pohyb s korekcí dráhy na str 172*, *EGMSetupLTAPP - Nastavit LTAPP protokol pro EGM na str 190* • Drobné korektury
C	Vydáno s verzí RobotWare 6.03. • Byla doplněna nová funkcionalita k instrukci *MotionProcessModeSet - Nastavit režim pohybového procesu na str 347*. • Byly doplněny instrukce, funkce a datové typy *CAP*. • Byly doplněny instrukce a funkce *Cyclic bool*. • Byly přidány instrukce a funkce ve vztahu k Functional Safety. • `signalxx` je nyní datový typ s poloviční hodnotou, který umožňuje operace orientované na hodnotu, viz *signalxx - Digitální a analogové signály na str 1582*. • Drobné korektury

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1 Instrukce

1.1 AccSet - Omezuje zrychlení

Použití

`AccSet` se používá při manipulaci s křehkými náklady. Umožňuje pomalejší zrychlení a zpomalení a výsledkem jsou jemnější pohyby robotu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `AccSet`:

Příklad 1

```
AccSet 50, 100;
```

Zrychlení je omezeno na 50 % normální hodnoty.

Příklad 2

```
AccSet 100, 50;
```

Rampa zrychlení je omezena na 50 % normální hodnoty.

Příklad 3

```
AccSet 100, 100 \FinePointRamp:=50;
```

Rampa zpomalení je omezena na 50 % normální hodnoty, když se provádí zpomalení směrem k jemnému bodu.

Argumenty

```
AccSet Acc Ramp [\FinePointRamp]
```

`Acc`

Datový typ: `num`

Zrychlení a zpomalení je procentuální část normálních hodnot. 100 % vyjadřuje maximální zrychlení. Maximální hodnota: 100%. Vstupní hodnota < 20 % udává 20 % maximálního zrychlení.

`Ramp`

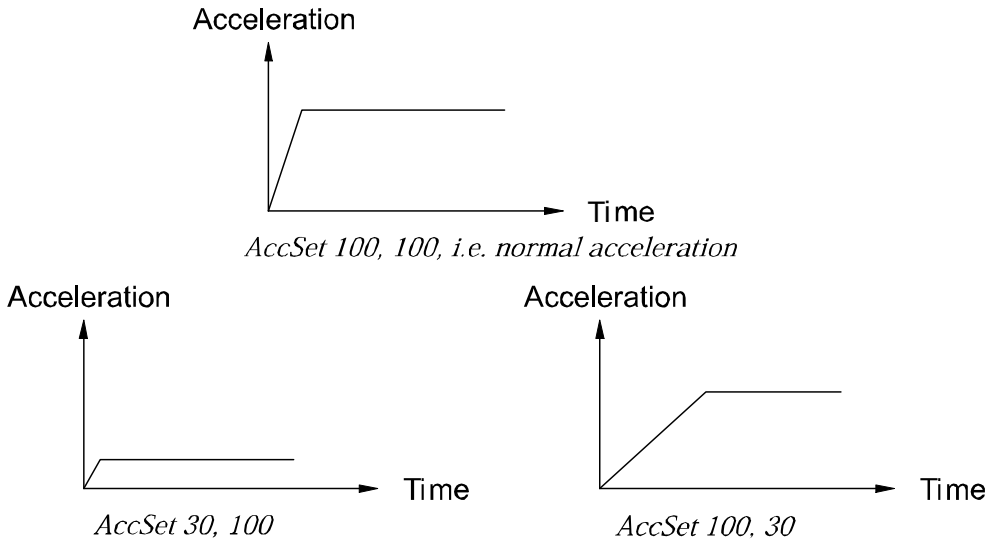
Datový typ: `num`

Rychlost, při které zrychlení a zpomalení narůstá jako procentuální část normálních hodnot. Ryv může být omezen snížením této hodnoty. 100 % vyjadřuje maximální rychlost. Maximální hodnota: 100%. Vstupní hodnota < 10 % udává 10 % maximální rychlosti.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Obrázky ukazují, že výsledkem snižování zrychlení jsou jemnější pohyby.



`[\FinePointRamp]`

Datový typ: `num`

Rychlost, při které zpomalení roste jako procentuální část normálních hodnot. Parametr ovlivňuje rampu pouze v případě, když robot zpomaluje ve směru k jemnému bodu. V jemném bodu je hodnota rampy zpomalení kombinací tohoto parametru a hodnoty `Ramp`, `Ramp` * `FinePointRamp`. **Parametr musí být větší než 0 a musí být v intervalu 0 až 100 %.**

Jestliže se tento volitelný argument nepoužije, hodnota `FinePointRamp` **se nastaví na výchozí hodnotu 100%.**

Vykonávání programu

Zrychlení se vztahuje jak k robotu, tak k externím osám, dokud není provedena nová instrukce `AccSet`.

Výchozí hodnoty (100 %) jsou nastavovány automaticky.

- **při používání restartovacího režimu Reset RAPID**
- **při načítání nového programu nebo nového modulu**
- **při spuštění vykonávání programu od začátku**
- **při posunu ukazatele programu k** `main`
- **při posunu ukazatele programu k rutině**
- **při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.**

Syntaxe

```
AccSet
  [ Acc ':=' ] < expression (IN) of num > ','
  [ Ramp ':=' ] < expression (IN) of num >
  [ '\'FinePointRamp ':=' < expression (IN) of num > ] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Kontrola zrychlení ve světovém souřadném systému	*WorldAccLim - Kontrolovat zrychlení ve světovém souřadném systému na str 978*
Omezit TCP zrychlení podél dráhy	*PathAccLim - Omezit TCP zrychlení podél dráhy na str 454*
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.2 ActEventBuffer - Aktivace vyrovnávací paměti pro události

Popis

`ActEventBuffer` se používá pro aktivaci použití bufferu událostí v programové úloze aktuálního pohybu.

Instrukce `ActEventBuffer` a `DeactEventBuffer` by se měly používat při kombinaci aplikace pomocí jemných bodů a pokračující aplikace, kde je nutné nastavovat signály předem kvůli pomalému procesnímu vybavení.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `ActEventBuffer`:

Příklad 1

```
...
DeactEventBuffer;
! Use an application that uses finepoints, such as SpotWelding
...
! Activate the event buffer again
ActEventBuffer;
! Now it is possible to use an application that needs
! to set signals in advance, such as Dispense
...
```

`DeactEventBuffer` deaktivuje konfigurovaný zásobník událostí. Při používání aplikace s jemnými body bude start robotu od jemného bodu rychlejší. Při aktivaci zásobníku událostí s `ActEventBuffer` je možné nastavovat signály dopředu pro aplikaci s pomalým procesním vybavením.

Vykonávání programu

Použití bufferu událostí se vztahuje k dalšímu provedenému libovolnému pohybu robotu a je platné až do provedení instrukce `DeactEventBuffer`.

Instrukce bude před aktivací zásobníku událostí čekat, až robot a externí osy dosáhnou stop bodu (`ToPoint` aktuální instrukce pohybu). Proto se doporučuje programovat pohybovou instrukci předcházející `ActEventBuffer` s jemným bodem.

Výchozí hodnota (použijte zásobník událostí = TRUE) se nastavuje automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

`ActEventBuffer` se nemůže provádět v RAPID rutině připojené k žádné z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart nebo Step.

Syntaxe

```
ActEventBuffer ';'
```

Související informace

Pro informace o	Viz
Deaktivace zásobníku událostí	*DeactEventBuffer - Deaktivace zásobníku událostí na str 148*
Konfigurace Event preset time	*Technická referenční příručka - Systémové parametry*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.3 ActUnit - Aktivuje mechanickou jednotku

Použití

`ActUnit` se používá pro aktivaci mechanické jednotky.

Může se používat pro určení, která jednotka bude aktivní, když se používají například společné pohonné jednotky.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `ActUnit`:

Příklad 1

```
ActUnit orbit_a;
```

Aktivace mechanické jednotky `orbit_a`.

Argumenty

```
ActUnit MechUnit
```

`MechUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky, která má být aktivována.

Vykonávání programu

Když jsou roboty a skutečná dráha externích os připraveny, dráha na aktuální úrovni dráhy je uvolněna a určená mechanická jednotka se aktivuje. To znamená, že je řízena a sledována robotem.

Jestliže několik mechanických jednotek sdílí společnou pohonnou jednotku, aktivace jedné z těchto mechanických jednotek připojí rovněž tuto jednotku ke společné pohonné jednotce.

Omezení

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (zonedata `fine`), nikoliv s průjezdným bodem, jinak nebude možný restart po selhání napájení.

`ActUnit` se nemůže provádět v RAPID rutině připojené k žádné z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart, Reset nebo Step.

Je možné používat `ActUnit` - `DeactUnit` na úrovni `StorePath`, ale stejné mechanické jednotky musí být aktivní při provádění `RestoPath` jako kdyby bylo provedeno `StorePath`. Taková operace na záznamníku dráhy a dráha na základní úrovni bude nedotčena, ale dráha na úrovni `StorePath` bude uvolněna.

Syntaxe

```
ActUnit
  [ MechUnit ':=' ] < variable (VAR) of mecunit> ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Deaktivace mechanických jednotek	*DeactUnit - Deaktivuje mechanickou jednotku na str 150*
Mechanické jednotky	*mecunit - Mechanická jednotka na str 1531*
Další příklady	*DeactUnit - Deaktivuje mechanickou jednotku na str 150*
Zkontrolujte, jestli je mechanická jednotka aktivována nebo nikoliv.	*IsMechUnitActive - Je mechanická jednotka aktivní na str 1211*
Záznamník dráhy	*PathRecMoveBwd - Posunout záznamník dráhy dozadu na str 458*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.4 Add - Přidává numerickou hodnotu

Použití

`Add` se používá pro přidání nebo odečtení hodnoty k nebo od numerické proměnné nebo perzistentu.

Základní příklady

Následující příklady názorně ukazují instrukci `Add`:

Příklad 1

```
Add reg1, 3;
```

`3` je přidáno k `reg1`, to znamená,`reg1:=reg1+3`.

Příklad 2

```
Add reg1, -reg2;
```

Hodnota `reg2` se odečítá od `reg1`, to znamená,`reg1:=reg1-reg2`.

Příklad 3

```
VAR dnum mydnum:=5;
Add mydnum, 500000000;
```

`500000000` je přidáno k `mydnum`, to znamená,mynum:=mynum+500000000.

Příklad 4

```
VAR dnum mydnum:=5000;
VAR num mynum:=6000;
Add mynum, DnumToNum(mydnum \Integer);
```

`5000` je přidáno k `mynum`, to znamená, `mynum:=mynum+5000`. Musíte použít `DnumToNum`, abyste dostali numerickou hodnotu `num`, kterou můžete použít společně s `num` proměnnou `mynum`.

Argumenty

```
Add Name | Dname AddValue | AddDvalue
```

`Name`

Datový typ: `num`

Jméno proměnné nebo perzistentu, které budou změněny.

`Dname`

Datový typ: `dnum`

Jméno proměnné nebo perzistentu, které budou změněny.

`AddValue`

Datový typ: `num`

Hodnota, která bude přidána.

`AddDvalue`

Datový typ: `dnum`

Hodnota, která bude přidána.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Jestliže hodnota, která bude přidána, je typu `dnum` a proměnná/perzistent, které by se měly změnit, je `num`, bude vydána chyba runtime. Kombinace argumentů není možná (viz Příklad 4 nahoře, jak toto vyřešit).

Syntaxe

```
Add
  [ Name ':=' ] < var or pers (INOUT) of num >
  | [ Dname ':=' ] < var or pers (INOUT) of dnum > ','
  [ AddValue ':=' ] < expression (IN) of num >
  | [ AddDvalue ':=' ] < expression (IN) of dnum > ';'
```

Související informace

Pro informace o	Viz
Přírůstkování proměnné o 1	*Incr - Přírůstky po 1 na str 251*
Snižování proměnné o 1	*Decr - Snižování po 1 na str 152*
Změna dat pomocí libovolného výrazu, například násobení	*":=" - Přiděluje hodnotu na str 33*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.5 AliasIO - Definovat V/V (I/O) signál se jménem aliasu

Použití

`AliasIO` se používá pro definování signálu jakéhokoliv typu se jménem aliasu nebo pro použití signálů ve vestavěných úlohových modulech.

Signály se jmény aliasu se mohou používat pro předdefinované generické programy bez jakékoliv modifikace programu před jeho během v různých robotických instalacích.

Instrukce `AliasIO` musí být provedena před jakýmkoliv použitím aktuálního signálu. Viz *Základní příklady na str 28* pro načtené moduly a *Další příklady na str 29* pro instalované moduly.

Základní příklady

Následující příklad názorně ukazuje instrukci `AliasIO`:

Viz také *Další příklady na str 29*.

Příklad 1

```
VAR signaldo alias_do;
PROC prog_start()
  AliasIO config_do, alias_do;
ENDPROC
```

Rutina `prog_start` je připojena k události `START` v systémových parametrech. Definující signál digitálního výstupu programu `alias_do` je připojen ke konfigurovanému signálu digitálního výstupu `config_do` při startu programu.

Argumenty

```
AliasIO FromSignal ToSignal
```

`FromSignal`

Datový typ: `signalxx` nebo `string`

Načtené moduly

Identifikátor signálu pojmenovaný podle konfigurace (datový typ `signalxx`), ze kterého ze zkopírován popisovač signálu. Signál se musí definovat v konfiguraci I/O.

Instalované moduly nebo načtené moduly:

Reference (`CONST`, `VAR` nebo jejich parametr) obsahující jméno signálu (datový typ `string`), ze kterého je zkopírován popisovač signálu po hledání v systému. Signál musí být definován v I/O konfiguraci.

`ToSignal`

Datový typ: `signalxx`

Identifikátor signálu podle programu (datový typ `signalxx`), do kterého je popisovač signálu zkopírován. Signál musí být deklarován v programu RAPID.

Stejný datový typ musí být použit (nebo nalezen) pro argumenty `FromSignal` a `ToSignal` a musí být jedním z typů `signalxx` (`signalai`, `signalao`, `signaldi`, `signaldo`, `signalgi`, nebo `signalgo`).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Hodnota popisovače signálu se zkopíruje ze signálu daného v argumentu `FromSignal` do signálu daného v argumentu `ToSignal`.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_ALIASIO_DEF`	`FromSignal` není definován v I/O konfiguraci nebo `ToSignal` není deklarován v programu RAPID nebo `ToSignal` není definován v I/O konfiguraci.
`ERR_ALIASIO_TYPE`	Datové typy pro argumenty `FromSignal` a `ToSignal` není stejný typ.
`ERR_NO_ALIASIO_DEF`	Proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

Další příklady

Více příkladů instrukce `AliasIO` je názorně uvedeno dole.

Příklad 1

```
VAR signaldi alias_di;
PROC prog_start()
  CONST string config_string := "config_di";
  AliasIO config_string, alias_di;
ENDPROC
```

Rutina `prog_start` je připojena k události `START` v systémových parametrech. Programem definovaný digitální vstupní signál `alias_di` je připojen ke konfigurovanému digitálnímu vstupnímu signálu `config_di` (přes konstantu `config_string`) při startu programu.

Omezení

Při startu programu nemůže být použit alias signál, dokud není provedena instrukce `AliasIO`.

Instrukce `AliasIO` **musí** být umístěna

- buď v rutině události provedené při startu programu (událost `START`)
- nebo v části programu prováděné po každém startu programu (před použitím signálu)

Kvůli zabránění chybám se nedoporučuje používat dynamické znovupřipojení signálu `AliasIO` k odlišným fyzickým signálům.

Syntaxe

```
AliasIO
  [ FromSignal ':=' ] < reference (REF) of anytype > ','
  [ ToSignal ':=' ] < variable (VAR) of anytype > ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.5 AliasIO - Definovat V/V (I/O) signál se jménem aliasu
RobotWare - OS
Pokračování

Související informace

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*
Definování rutin událostí	*Technická referenční příručka - Systémové parametry*
Načtené/instalované úlohové moduly	*Technická referenční příručka - Systémové parametry*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.6 AliasIOReset - Resetování I/O signálu s alias jménem

Použití

`AliasIOReset` se používá pro reset signálu, který byl použit v předchozím volání na `AliasIO`.

Základní příklady

Následující příklad názorně ukazuje instrukci `AliasIOReset`:

Příklad 1

```
VAR signaldo alias_do;
PROC myproc()
  AliasIO config_do, alias_do;
  SetDO alias_do, 1;
  ..
  AliasIOReset alias_do;
ENDPROC
```

Programem definovaný digitální výstupní signál `alias_do` je připojen ke konfigurovanému digitálnímu výstupnímu signálu `config_do` na začátku procedury `myproc`. Signál `config_do` je definován v konfiguraci I/O. Později, když už by se neměl používat `alias_do`, se spojení aliasu odstraní.

Argumenty

```
AliasIOReset Signal
```

`Signal`

Datový typ: `signalxx`

Identifikátor signálu podle programu (datový typ `signalxx`), který by měl být resetován. Signál musí být deklarován v programu RAPID.

Vykonávání programu

Celé spojení alias je odstraněno. Signál se nemůže použít, dokud není vytvořeno nové alias spojení s `AliasIO`.

Omezení

Signály, které jsou definovány v konfiguraci I/O, není možné resetovat. Použít se mohou pouze signály, které byly použity v instrukci `AliasIO` a jsou deklarovány v programu RAPID.

Syntaxe

```
AliasIOReset
  [ Signal ':=' ] < variable (VAR) of anytype > ';'
```

Související informace

Pro informace o	Viz
Definovat I/O signál se jménem aliasu	*AliasIO - Definovat V/V (I/O) signál se jménem aliasu na str 28*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.6 AliasIOReset - Resetování I/O signálu s alias jménem

RobotWare - OS

Pokračování

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*
Definování rutin událostí	*Technická referenční příručka - Systémové parametry*
Načtené/instalované úlohové moduly	*Technická referenční příručka - Systémové parametry*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.7 ":=" - Přiděluje hodnotu

Použití

Instrukce “:=” se používá pro přidělení nové hodnoty k datům. Tato hodnota může být jakákoliv od konstantní hodnoty k aritmetickému výraz, například `reg1+5*reg3`.

Základní příklady

Následující příklady názorně ukazují instrukci “:=”:

Viz také *Další příklady na str 33*.

Příklad 1

```
reg1 := 5;
```

`reg1` má přidělenu hodnotu `5`.

Příklad 2

```
reg1 := reg2 - reg3;
```

`reg1` má přidělenu hodnotu, kterou výpočet `reg2-reg3` vrací.

Příklad 3

```
counter := counter + 1;
```

`counter` je přírůstkován po jedné.

Argumenty

```
Data := Value
```

`Data`

Datový typ: `All`

Data, kterým bude přidělena nová hodnota.

`Value`

Datový typ: `Same as Data`

Požadovaná hodnota.

Další příklady

Více příkladů instrukce “:=” je uvedeno dole.

Příklad 1

```
tool1.tframe.trans.x := tool1.tframe.trans.x + 20;
```

TCP pro `tool1` je posunuto o `20` mm ve směru X.

Příklad 2

```
pallet{5,8} := Abs(value);
```

Prvku v matrici `pallet` je přidělena hodnota, která se rovná absolutní hodnotě proměnné `value`.

Omezení

Data (jejichž hodnota se změní) nesmějí být

- konstanta
- nehodnotový datový typ.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.7 ":=" - Přiděluje hodnotu

RobotWare - OS

Pokračování

Data a hodnota musí mít podobné (stejné nebo alias) datové typy.

Syntaxe

```
<assignment target> ':=' <expression> ';'
```

Související informace

Pro informace o	Viz
Výrazy	*Technická referenční příručka - Přehled RAPID*
Nehodnotové datové typy	*Technická referenční příručka - Přehled RAPID*
Přidělování prvotní hodnoty datům	*Návod k použití - IRC5 s jednotkou FlexPendant*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.8 BitClear - Vyčistit určený bit v bytu nebo dnum datech

Použití

`BitClear` se používá pro vyčištění (nastavení na 0) určeného bitu v určených datech `byte` nebo `dnum`.

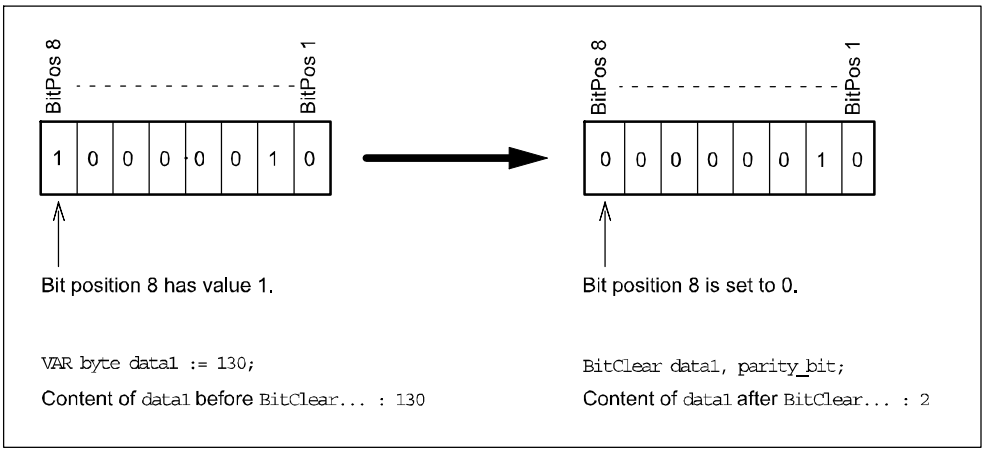
Základní příklady

Následující příklady názorně ukazují instrukci `BitClear`:

Příklad 1

```
CONST num parity_bit := 8;
VAR byte data1 := 130;
BitClear data1, parity_bit;
```

Bit číslo `8` (parity_bit) v proměnné `data1` bude nastaven na 0, například, obsah proměnné `data1` bude změněn ze 130 na 2 (zastoupení celého čísla). Bitová manipulace datového typu `byte` při použití `BitClear` je uvedena na následujícím obrázku.



xx0500002147

Příklad 2

```
CONST num parity_bit := 52;
VAR dnum data2 := 2251799813685378;
BitClear data2, parity_bit;
```

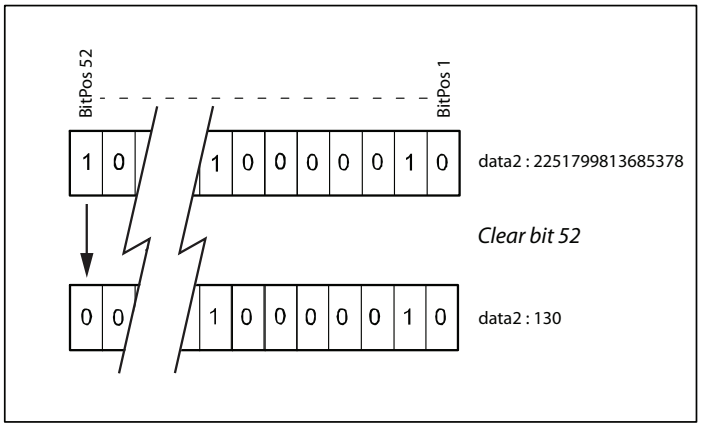
Bit číslo `52` (parity_bit) v proměnné `data2` bude nastaven na 0, např. obsah proměnné `data2` bude změněn z 2251799813685378 na 130 (zastoupení celého

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.8 BitClear - Vyčistit určený bit v bytu nebo dnum datech
RobotWare - OS
Pokračování

čísla). Bitová manipulace datového typu `dnum` při použití `BitClear` je uvedena na obrázku dole.



xx1200000014

Argumenty

```
BitClear BitData | DnumData BitPos
```

`BitData`

Datový typ: `byte`

Bitová data, v zastoupení celého čísla, která budou změněna.

`DnumData`

Datový typ: `dnum`

Bitová data dnum, v zastoupení celého čísla, která budou změněna.

`BitPos`

Bit Position

Datový typ: `num`

Pozice bitu (1-8) v `BitData`, nebo pozice bitu (1-52) v `DnumData`, bude nastavena na 0.

Omezení

Rozsah pro datový typ je `byte` 0 - 255 desítkový.

Pozice bitu je platná od 1 - 8 pro datový typ `byte`.

Rozsah pro datový typ je `dnum` 0 - 4503599627370495 desítkový.

Pozice bitu je platná od 1 - 52 pro datový typ `dnum`.

Syntaxe

```
BitClear
  [ BitData ':=' ] < var or pers (INOUT) of byte >
  | [ DnumData ':=' ] < var or pers (INOUT) of dnum > ','
  [ BitPos ':=' ] < expression (IN) of num > ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Nastavit určený bit do dat byte nebo dnum	*BitSet - Nastavit určený bit do dat byte nebo dnum na str 38*
Zkontrolujte, jestli je nastaven určený bit v datech byte	*BitCheck - Zkontrolujte, jestli je nastaven určený bit v datech byte na str 1052*
Zkontrolujte, jestli je nastaven určený bit v datech dnum	*BitCheckDnum - Zkontrolujte, jestli je nastaven určený bit v datech dnum na str 1054*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.9 BitSet - Nastavit určený bit do dat byte nebo dnum

Použití

`BitSet` se používá pro nastavení určeného bitu na 1 v definovaných datech `byte` nebo `dnum`.

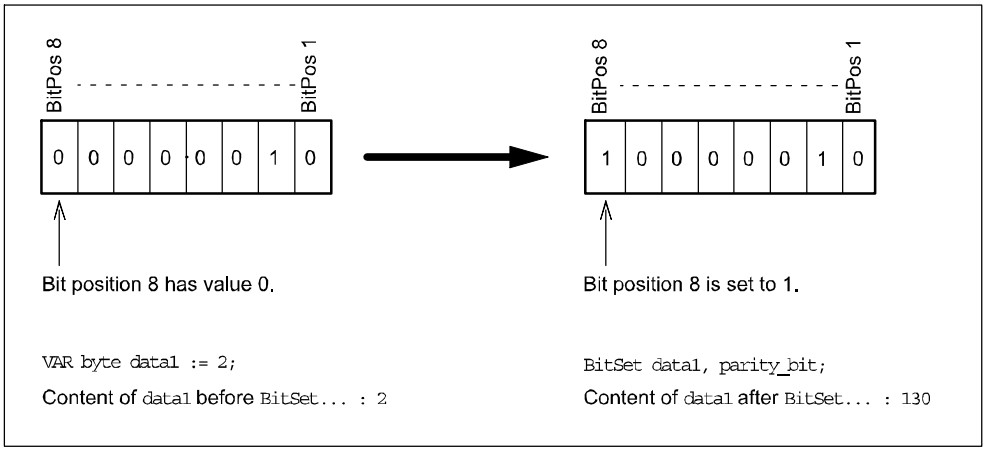
Základní příklady

Následující příklady názorně ukazují instrukci `BitSet`:

Příklad 1

```
CONST num parity_bit := 8;
VAR byte data1 := 2;
BitSet data1, parity_bit;
```

Bit číslo 8 (`parity_bit`) v proměnné `data1` bude nastaven na `1`, například, obsah proměnné `data1` bude změněn z 2 na 130 (zastoupení celého čísla). Bitová manipulace datového typu `byte` při použití `BitSet` je uvedena na obrázku dole.



xx0500002148

Příklad 2

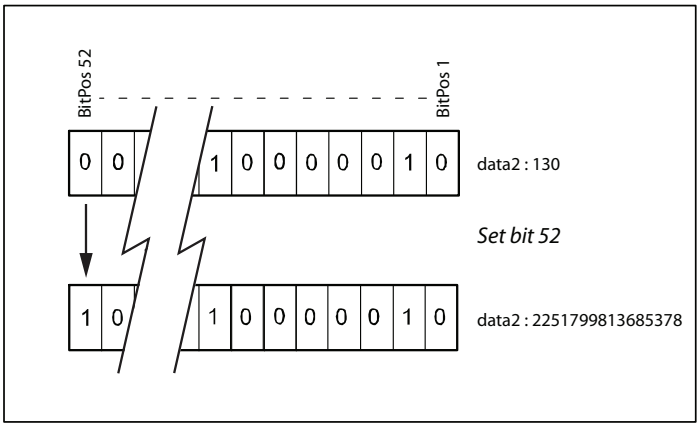
```
CONST num parity_bit := 52;
VAR dnum data2 := 130;
BitSet data2, parity_bit;
```

Bit číslo 52 (`parity_bit`) v proměnné `data2` bude nastaven na `1`, např. obsah proměnné `data2` bude změněn ze 130 na 2251799813685378 (zastoupení celého

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

čísla). Bitová manipulace datového typu `dnum` při použití `BitSet` je uvedena na obrázku dole.



xx1200000015

Argumenty

```
BitSet BitData | DnumData BitPos
```

`BitData`

Datový typ: `byte`

Bitová data, v zastoupení celého čísla, která budou změněna.

`DnumData`

Datový typ: `dnum`

Bitová data, v zastoupení celého čísla, která budou změněna.

`BitPos`

Bit Position

Datový typ: `num`

Pozice bitu (1-8) v `BitData`, nebo pozice bitu (1-52) v `DnumData`, bude nastavena na 1.

Omezení

Rozsah pro datový typ je `byte` celé číslo 0 - 255.

Pozice bitu je platná od 1 - 8 pro datový typ `byte`.

Rozsah pro datový typ je `dnum` celé číslo 0 - 4503599627370495.

Pozice bitu je platná od 1 - 52 pro datový typ `dnum`.

Syntaxe

```
BitSet
  [ BitData':=' ] < var or pers (INOUT) of byte >
  | [ DnumData':=' ] < var or pers (INOUT) of dnum > ','
  [ BitPos':=' ] < expression (IN) of num > ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.9 BitSet - Nastavit určený bit do dat byte nebo dnum

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Vyčistit určený bit v datech byte nebo dnum	*BitClear - Vyčistit určený bit v bytu nebo dnum datech na str 35*
Zkontrolujte, jestli je nastaven určený bit v datech byte	*BitCheck - Zkontrolujte, jestli je nastaven určený bit v datech byte na str 1052*
Zkontrolujte, jestli je nastaven určený bit v datech dnum	*BitCheckDnum - Zkontrolujte, jestli je nastaven určený bit v datech dnum na str 1054*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.10 BookErrNo - Zapsat chybové číslo systému RAPID

Použití

`BookErrNo` se používá k zapsání nového chybového čísla systému RAPID.

Základní příklady

Následující příklad názorně ukazuje instrukci `BookErrNo`:

Příklad 1

```
! Introduce a new error number in a glue system
! Note: The new error variable must be declared with the initial
     value -1
VAR errnum ERR_GLUEFLOW := -1;

! Book the new RAPID system error number
BookErrNo ERR_GLUEFLOW;
```

Proměnná `ERR_GLUEFLOW` bude přiřazena volnému chybovému číslu systému pro použití v kódu RAPID.

```
! Use the new error number
IF di1 = 0 THEN
  RAISE ERR_GLUEFLOW;
ELSE
...
ENDIF

! Error handling
ERROR
  IF ERRNO = ERR_GLUEFLOW THEN
    ...
  ELSE
    ...
  ENDIF
```

Jestliže digitální vstup `di1` je `0`, nové zapsané chybové číslo bude zavedeno a proměnná systémové chyby `ERRNO` bude nastavena na nově zapsané číslo chyby. Řešení uživatelem vytvořených chyb se potom bude provádět jako obvykle v obslužném programu pro chyby.

Argumenty

```
BookErrNo ErrorName
```

`ErrorName`

Datový typ: `errnum`

Jméno proměnné nové systémové chyby RAPID.

Omezení

Proměnná nové chyby nesmí být deklarována jako proměnná rutiny.

Proměnná nové chyby musí být deklarována s počáteční hodnotou -1, která dává informaci, že tato chyba by měla být systémovou chybou RAPID.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.10 BookErrNo - Zapsat chybové číslo systému RAPID

RobotWare - OS

Pokračování

Syntaxe

```
BookErrNo
  [ ErrorName ':='] < variable (VAR) of errnum > ';'
```

Související informace

Pro informace o	Viz
Řešení chyb	*Technická referenční příručka - Přehled RAPID*
Číslo chyby	*errnum - Chybové číslo na str 1495*
Volat obslužnou rutinu chyb	*RAISE - Volá chybový handler na str 512*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.11 Break - Přerušení provádění programu

Použití

`Break` se používá k provedení okamžitého přerušení provádění programu pro účely ladění programového kódu RAPID. Pohyb robotu se náhle zastaví.

Základní příklady

Následující příklad názorně ukazuje instrukci `Break`:

Příklad 1

```
...
Break;
...
```

Provádění programu se zastaví a je možné analyzovat proměnné, hodnoty atd. pro účely odladění programu.

Vykonávání programu

Instrukce náhle zastaví provádění programu bez čekání na robot a externí osy, až dosáhnou svých naprogramovaných bodů určení pro právě prováděný pohyb. Provádění programu může být potom obnoveno od další instrukce.

Jestliže je v některé události rutiny instrukce `Break` , provádění rutiny bude přerušeno a žádná událost STOP rutiny nebude provedena. Událost rutiny bude provedena od začátku příště, až stejná událost nastane.

Syntaxe

```
Break';'
```

Související informace

Pro informace o	Viz
Zastavení pro činnosti programu	*Stop - Ukončuje vykonávání programu na str 731*
Zastavení po fatální chybě	*EXIT - Ukončuje vykonávání programu na str 213*
Ukončení provádění programu	*EXIT - Ukončuje vykonávání programu na str 213*
Pouze zastavení pohybů robotu	*StopMove - Zastavuje pohyby robotu na str 736*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.12 CallByVar - Volat proceduru přes proměnnou

Použití

`CallByVar` (*Call By Variable*) se může používat pro volání procedur s konkrétními jmény, například`proc_name1, proc_name2, proc_name3 ... proc_namex` přes proměnnou.

Základní příklady

Následující příklad názorně ukazuje instrukci `CallByVar`:

Viz také *Další příklady na str 44*.

Příklad 1

```
reg1 := 2;
CallByVar "proc", reg1;
```

Procedura `proc2` je volána.

Argumenty

```
CallByVar Name Number
```

`Name`

Datový typ: `string`

První část jména procedury, například `proc_name`.

`Number`

Datový typ: `num`

Numerická hodnota pro číslo procedury. Tato hodnota bude převedena na řetězec a udává druhou část jména procedury, například `1`. Hodnota musí být kladné celé číslo.

Další příklady

Další příklady, jak vytvořit statický a dynamický výběr volání rutiny.

Příklad 1 - Statický výběr volání procedury

```
TEST reg1
  CASE 1:
    lf_door door_loc;
  CASE 2:
    rf_door door_loc;
  CASE 3:
    lr_door door_loc;
  CASE 4:
    rr_door door_loc;
  DEFAULT:
    EXIT;
ENDTEST
```

V závislosti na tom, jestli hodnota registru `reg1` je 1, 2, 3, nebo 4, jsou volány různé procedury, které provádějí příslušný typ práce pro vybrané dveře. Umístění dveří v argumentu `door_loc`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Příklad 2 - Dynamický výběr volání procedury se syntaxí RAPID

```
reg1 := 2;
%"proc"+NumToStr(reg1,0)% door_loc;
```

Procedura `proc2` **je volána s argumentem** `door_loc`.

Omezení: Všechny procedury musí mít specifické jméno, například `proc1, proc2, proc3`.

Příklad 3 - Dynamický výběr volání procedury s CallByVar

```
reg1 := 2;
CallByVar "proc",reg1;
```

Procedura `proc2` **je volána.**

Omezení: Všechny procedury musí mít specifické jméno, například `proc1`, `proc2`, `proc3` **a nemohou být použity žádné argumenty.**

Omezení

Může se používat pouze pro volání procedur bez parametrů.

Nemůže se používat pro volání LOCAL procedur.

Provedení `CallByVar` **trvá o trochu déle než provedení normálního volání procedury.**

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` **bude nastavena na:**

Název	Příčina chyby
ERR_ARGVALERR	Argument `Number` je < 0 nebo to není celé číslo.
ERR_REFUNKPRC	Reference směřuje k neznámé proceduře.
ERR_CALLPROC	Chyba volání procedury (není procedura).

Syntaxe

```
CallByVar
  [Name ':='] <expression (IN) of string>','
  [Number ':='] <expression (IN) of num>';'
```

Související informace

Pro informace o	Viz
Volání procedur	*Technická referenční příručka - Přehled RAPID* *Návod k použití - IRC5 s jednotkou FlexPendant*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.13 CamFlush - Odstraňuje sběrná data pro kameru

Použití

`CamFlush` se používá pro pročištění (odstranění) sběrných dat `cameratarget` pro kameru.

Základní příklady

Následující příklad názorně ukazuje instrukci `CamFlush`.

Příklad 1

```
CamFlush mycamera;
```

Sběrná data pro kameru `mycamera` byla odstraněna.

Argumenty

```
CamFlush Camera
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

Syntaxe

```
CamFlush
  [ Camera ':=' ] < variable (VAR) of cameradev > ';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.14 CamGetParameter - Získat různé jmenovité parametry kamery

Použití

`CamGetParameter` se používá pro získání jmenovitých parametrů, které může kamera odhalit. Uživatel musí znát jméno parametru a jeho vratný typ, aby bylo možné získat jeho hodnotu.

Základní příklady

Následující příklad názorně ukazuje instrukci `CamGetParameter`.

Příklad 1

```
VAR bool mybool:=FALSE;
...
CamGetParameter mycamera, "Pattern_1.Tool_Enabled_Status"
     \BoolVar:=mybool;
TPWite "The current value of Pattern_1.Tool_Enabled_Status is: "
     \Bool:=mybool;
```

Získat jmenovitý booleánský parametr `Pattern_1.Tool_Enabled_Status` a zapsat hodnotu na FlexPendant.

Argumenty

```
CamGetParameter Camera ParName [\Num] | [\Bool] | [\Str]
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

`ParName`

Parameter Name

Datový typ: `string`

Jméno parametru v kameře.

`[\NumVar]`

Datový typ: `num`

Proměnná (**VAR**) pro uložení numerické hodnoty získaného datového objektu.

`[\BoolVar]`

Datový typ: `bool`

Proměnná (**VAR**) pro uložení booleánské hodnoty získaného datového objektu.

`[\StrVar]`

Datový typ: `string`

Proměnná (**VAR**) pro uložení hodnoty řetězce získaného datového objektu.

Vykonávání programu

Instrukce čte stanovený parametr přímo, když je instrukce prováděna a vrací hodnotu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.14 CamGetParameter - Získat různé jmenovité parametry kamery

Integrated Vision
Pokračování

Jestliže se instrukce používá pro čtení výsledku z analýzy obrazu, zajistěte, aby kamera dokončila zpracování obrazu před získáváním dat.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_CAM_BUSY	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.
ERR_CAM_COM_TIMEOUT	Chyba komunikace s kamerou. Kamera je pravděpodobně odpojena.
ERR_CAM_GET_MISMATCH	Parametr získaný z kamery s instrukcí CamGetParameter má nesprávný datový typ.

Syntaxe

```
CamGetParameter
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ ParName ':=' ] < expression (IN) of string >
  [ '\'NumVar ':=' < variable (VAR) of num > ]
  | [ '\'BoolVar ':=' < variable (VAR) of bool > ]
  | [ '\'StrVar ':=' < variable (VAR) of string > ] ';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.15 CamGetResult - Získává cíl kamery ze sběrných dat

Použití

`CamGetResult` *(Camera Get Result)* **se používá pro získání cíle kamery ze sbírky výsledků vidění.**

Základní příklady

Následující příklad názorně ukazuje instrukci `CamGetResult`.

Příklad 1

```
VAR num mysceneid;
VAR cameratarget mycamtarget;
...
CamReqImage mycamera \SceneId:= mysceneid;
CamGetResult mycamera, mycamtarget \SceneId:= mysceneid;
```

Přikázat kameře `mycamera` **získat obraz. Získat výsledek vidění vznikající z obrazu s** `SceneId`.

Argumenty

```
CamGetResult Camera CamTarget [\SceneId] [\MaxTime]
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

`CamTarget`

Cíl kamery

Datový typ: `cameratarget`

Proměnná, kde bude výsledek vidění uložen.

`[\SceneId]`

Identifikace scény

Datový typ: `num`

`SceneId` **je identifikátor, který stanoví, od kterého obrazu byl generován** `cameratarget`.

`[\MaxTime]`

Maximální čas

Datový typ: `num`

Maximální množství času v sekundách, kdy provedení programu čeká. Maximální přípustná hodnota je 120 sekund.

Vykonávání programu

`CamGetResult` **získává cíl kamery ze sbírky výsledků vidění. Jestliže není použit žádný** `SceneId` **nebo** `MaxTime` **a není žádný výsledek k získání, instrukce skončí natrvalo. Jestliže se použije** `SceneId` **v** `CamGetResult`**, mělo by to být generováno v předchozí instrukci** `CamReqImage`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.15 CamGetResult - Získává cíl kamery ze sběrných dat

Integrated Vision
Pokračování

`SceneId` se může používat pouze v případě, jestliže obraz byl přikázán od instrukce `CamReqImage`. Jestliže jsou obrazy generovány externím I/O signálem, `SceneId` nemůže být použit v instrukci `CamGetResult`.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_CAM_BUSY`	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.
`ERR_CAM_MAXTIME`	Ve stanoveném čase nebyl získán žádný výsledek.
`ERR_CAM_NO_MORE_DATA`	Žádné další vizuální výsledky nemohly být získány pro použitý `SceneId`, nebo během stanoveného času nemohl být získán žádný vizuální výsledek.

Syntaxe

```
CamGetResult
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ CamTarget ':=' ] < variable (VAR) of CameraTarget >
  [ '\'SceneId ':=' < expression (IN) of num > ]
  [ '\'MaxTime ':=' < expression (IN) of num > ] ';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.16 CamLoadJob - Načíst kamerovou úlohu do kamery

Použití

`CamLoadJob` *(Camera Load Job)* načítá úlohu pro kameru, *job*, popisuje parametry expozice, kalibraci a které vizuální nástroje aplikovat.

Základní příklady

Následující příklad názorně ukazuje instrukci `CamLoadJob`.

Příklad 1

```
CamSetProgramMode mycamera;
CamLoadJob mycamera, "myjob.job";
CamSetRunMode mycamera;
```

Práce `myjob` je načtena do kamery jméno `mycamera`.

Argumenty

```
CamLoadJob Camera JobName [\KeepTargets] [\MaxTime]
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

`Name`

Datový typ: `string`

Jméno práce k načtení do kamery.

`[\KeepTargets]`

Datový typ: `switch`

Tento argument se používá ke stanovení, jestli některé existující cíle kamery vytvořené kamerou by měly být zachovány.

`[\MaxTime]`

Datový typ: `num`

Maximální množství času v sekundách, kdy provedení programu čeká. Maximální přípustná hodnota je 120 sekund.

Vykonávání programu

Provedení `CamLoadJob` bude čekat, až je práce načtena nebo selže s chybou vypršení času. Když je použit volitelný argument `KeepTargets`, starý sběr dat pro konkrétní kameru je zachován. Výchozím chováním je odstranit (vyčistit) starý sběr dat.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_CAM_BUSY`	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.16 CamLoadJob - Načíst kamerovou úlohu do kamery

Integrated Vision
Pokračování

Název	Příčina chyby
ERR_CAM_COM_TIMEOUT	Chyba komunikace s kamerou. Kamera je pravděpodobně odpojena.
ERR_CAM_MAXTIME	Práce pro kameru nebyla načtena během určeného času.
ERR_CAM_NO_PROGMODE	Kamera není v programovém režimu

Omezení

Je možné pouze provést `CamLoadJob`, když je kamera nastavena v programovém režimu. Použijte instrukci `CamSetProgramMode` pro nastavení kamery do programového režimu.

Aby bylo možné načíst práci, soubor práce musí být uložen na flash disku kamery.

Syntaxe

```
CamLoadJob
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ JobName ':=' ] <expression (IN) of string >
  [ '\'KeepTargets ]
  [ '\'MaxTime ':=' <expression (IN) of num>]';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.17 CamReqImage - Přikázat kameře získat obraz

Použití

`CamReqImage` ***(Camera Request Image)*** **přikazuje kameře získat obraz.**

Základní příklady

Následující příklad názorně ukazuje instrukci `CamReqImage`**.**

Příklad 1

```
CamReqImage mycamera;
```

Přikázat kameře `mycamera` **získat nový obraz.**

Argumenty

```
CamReqImage Camera [\SceneId] [\KeepTargets] [\AwaitComplete]
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

`[\SceneId]`

Identifikace scény

Datový typ: `num`

Volitelný argument `SceneId` **je identifikátor pro získaný obraz. Je generován pro každý provedený** `CamReqImage` **pomocí volitelného argumentu** `SceneId`**. Identifikátor je celé číslo mezi 1 a 8388608. Jestliže není použit žádný** `SceneId`**, hodnota identifikátoru se nastaví na 0.**

`[\KeepTargets]`

Datový typ: `switch`

Tento argument se používá ke stanovení, jestli starý sběr dat pro určenou kameru má být zachován.

`[\AwaitComplete]`

Datový typ: `switch`

Jestliže je stanoven volitelný argument `\AwaitComplete`**, instrukce čeká, dokud nebudou přijaty výsledky z obrazu. Jestliže nebyly generovány žádné výsledky, například proto, že v obraze nebyl žádný díl, bude vyvolána chyba** `ERR_CAM_REQ_IMAGE`**.**

Když je použit `\AwaitComplete` **typ spouště kamery musí být nastaven na Externí.**

Vykonávání programu

`CamReqImage` **přikazuje určené kameře získat obraz. Jestliže je použit volitelný argument** `SceneId`**, dostupné vizuální výsledky získaného obrazu jsou označeny jedinečným číslem generovaným instrukcí.**

Jestliže je použit volitelný argument `KeepTargets`**, je zachován starý sběr dat pro určenou kameru. Výchozí chování je odstranit (vyčistit) veškerý starý sběr dat.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.17 CamReqImage - Přikázat kameře získat obraz

Integrated Vision
Pokračování

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_CAM_BUSY	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.
ERR_CAM_COM_TIMEOUT	Chyba komunikace s kamerou. Kamera je pravděpodobně odpojena.
ERR_CAM_NO_RUNMODE	Kamera není v režimu běhu
ERR_CAM_REQ_IMAGE	Kamera nemohla generovat žádný výsledek z obrazu.

Omezení

Je možné provést CamReqImage, jen když je kamera nastavena v režimu běhu. Použijte instrukci CamSetRunMode pro nastavení kamery do režimu běhu.

Syntaxe

```
CamReqImage
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ '\'SceneId ':=' < variable (VAR) of num > ]
  [ '\'KeepTargets ]
  [ '\'AwaitComplete ]';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.18 CamSetExposure - Nastavit data podle kamery

Použití

`CamSetExposure` (*Camera Set Exposure*) nastavuje data podle kamery a umožňuje adaptovat parametry obrazu podle světelných podmínek okolí.

Základní příklady

Následující příklad názorně ukazuje instrukci `CamSetExposure`.

Příklad 1

```
CamSetExposure mycamera \ExposureTime:=10;
```

Přikázat kameře `mycamera` změnit čas expozice na 10 ms.

Argumenty

```
CamSetExposure Camera [\ExposureTime] [\Brightness] [\Contrast]
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

`[\ExposureTime]`

Datový typ: `num`

Jestliže je použit tento volitelný argument, expoziční čas kamery je aktualizován. Hodnota je v milisekundách (ms).

`[\Brightness]`

Datový typ: `num`

Jestliže je použit tento volitelný argument, nastavení jasu kamery je aktualizováno. Hodnota se normálně vyjadřuje na stupnici 0 až 1.

`[\Contrast]`

Datový typ: `num`

Jestliže je použit tento volitelný argument, nastavení kontrastu kamery je aktualizováno. Hodnota se normálně vyjadřuje na stupnici 0 až 1.

Vykonávání programu

Instrukce aktualizuje čas expozice, jas a kontrast, jestliže je toto možné aktualizovat na určené kameře. Jestliže kamera nepodporuje toto nastavení, bude vydána chybová zpráva uživateli a provádění programu se zastaví.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_CAM_COM_TIMEOUT`	Chyba komunikace s kamerou. Kamera je pravděpodobně odpojena.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.18 CamSetExposure - Nastavit data podle kamery

Integrated Vision

Pokračování

Syntaxe

```
CamSetExposure
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ '\'ExposureTime ':=' < variable (IN) of num > ]
  [ '\'Brightness ':=' < variable (IN) of num > ]
  [ '\'Contrast ':=' < variable (IN) of num > ] ';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.19 CamSetParameter - Nastavit různé jmenovité parametry kamery

Použití

`CamSetParameter` se používá pro nastavení různých jmenovitých parametrů kamery, které může kamera exponovat. S touto instrukcí je možné měnit různé parametry v kameře při běhu. Uživatel musí znát jméno parametru a jeho typ kvůli nastavení jeho hodnoty.

Základní příklady

Následující příklad názorně ukazuje instrukci `CamSetParameter`.

Příklad 1

```
CamSetParameter mycamera, "Pattern_1.Tool_Enabled" \BoolVal:=FALSE;
CamSetRunMode mycamera;
```

V tomto příkladu je parametr nazvaný "`Pattern_1.Tool_Enabled`" nastaven na False, což znamená, že určený nástroj vizualizace by neměl být proveden, když je získán obraz.

To umožní rychlejší provedení vizualizačního nástroje. Nicméně, nástroj stále produkuje výsledky s hodnotami z posledního aktivního provedení. Aby se nepoužívaly tyto cíle, vyřaďte je z programu RAPID.

Argumenty

```
CamSetParameter Camera ParName [\Num] | [\Bool] | [\Str]
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

`ParName`

Datový typ: `string`

Jméno parametru v kameře.

`[\NumVal]`

Datový typ: `num`

Numerická hodnota k nastavení pro parametr kamery se jménem nastaveným v argumentu `ParName`.

`[\BoolVal]`

Datový typ: `bool`

Booleánská hodnota k nastavení pro parametr kamery se jménem nastaveným v argumentu `ParName`.

`[\StrVal]`

Datový typ: `string`

Řetězcová hodnota k nastavení pro parametr kamery se jménem nastaveným v argumentu `ParName`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.19 CamSetParameter - Nastavit různé jmenovité parametry kamery

Integrated Vision
Pokračování

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_CAM_BUSY	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.
ERR_CAM_COM_TIMEOUT	Chyba komunikace s kamerou. Kamera je pravděpodobně odpojena.
ERR_CAM_SET_MISMATCH	Parametr zapsaný do kamery s instrukcí CamSetParameter má špatný datový typ nebo hodnota je mimo rozsah.

Syntaxe

```
CamSetParameter
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ ParName ':=' ] < expression (IN) of string >
  [ '\'NumVal ':=' < expression (IN) of num > ]
  | [ '\'BoolVal ':=' < expression (IN) of bool > ]
  | [ '\'StrVal ':=' < expression (IN) of string > ] ';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.20 CamSetProgramMode - Přikazuje kameře přejít do programového režimu

Použití

`CamSetProgramMode` *(Camera Set Program Mode)* přikazuje kameře přejít do programového režimu (offline).

Základní příklady

Následující příklad názorně ukazuje instrukci `CamSetProgramMode`.

Příklad 1

```
CamSetProgramMode mycamera;
CamLoadJob mycamera, "myjob.job";
CamSetRunMode mycamera;
...
```

Nejprve změňte kameru do programovacího režimu. Potom načtěte `myjob` do kamery. Potom přikažte kameře přejít to režimu běhu.

Argumenty

```
CamSetProgramMode Camera
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

Vykonávání programu

Při přikazování kamery k přechodu do programového režimu s instrukcí `CamSetProgramMode`, bude možné změnit nastavení a načíst práci do kamery.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_CAM_BUSY`	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.
`ERR_CAM_COM_TIMEOUT`	Chyba komunikace s kamerou. Kamera je pravděpodobně odpojena.

Syntaxe

```
CamSetProgramMode
  [ Camera ':=' ] < variable (VAR) of cameradev > ';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.21 CamSetRunMode - Přikazuje kameře přejít do režimu běhu

Použití

`CamSetRunMode` *(Camera Set Running Mode)* přikazuje kameře přejít do režimu běhu (online) a aktualizuje řadič na aktuálním výstupu na konfiguraci RAPID.

Základní příklady

Následující příklad názorně ukazuje instrukci `CamSetRunMode`.

Příklad 1

```
CamSetProgramMode mycamera;
CamLoadJob mycamera, "myjob.job";
...
CamSetRunMode mycamera;
```

Nejprve změňte kameru do programovacího režimu. Potom načtěte `myjob` do kamery. Potom přikažte kameře přejít do režimu běhu s instrukcí `CamSetRunMode`.

Argumenty

```
CamSetRunMode Camera
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

Vykonávání programu

Při přikazování kamery přejít do režimu běhu s `CamSetRunMode` je možné začít snímat obrazy.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_CAM_BUSY`	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.
`ERR_CAM_COM_TIMEOUT`	Chyba komunikace s kamerou. Kamera je pravděpodobně odpojena.

Syntaxe

```
CamSetRunMode
  [ Camera ':=' ] < variable (VAR) of cameradev > ';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.22 CamStartLoadJob - Začít načítat kamerovou úlohu do kamery

Použití

`CamStartLoadJob` začne načítat práci do kamery a potom bude provádění pokračovat na další instrukci. Když načítání pokračuje, jiné instrukce je možné provádět současně.

Základní příklady

Následující příklad názorně ukazuje instrukci `CamStartLoadJob`.

Příklad 1

```
...
CamStartLoadJob mycamera, "myjob.job";
MoveL p1, v1000, fine, tool2;
CamWaitLoadJob mycamera;
CamSetRunMode mycamera;
CamReqImage mycamera;
...
```

Nejdříve je spuštěno načítání práce do kamery, a zatímco načítání probíhá, je proveden pohyb do pozice p1. Když je pohyb připraven a načítání je u konce, je získán obraz.

Argumenty

```
CamStartLoadJob Camera Name [\KeepTargets]
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

`Name`

Datový typ: `string`

Jméno práce k načtení do kamery.

`[\KeepTargets]`

Datový typ: `switch`

Tento argument se používá ke stanovení, jestli starý sběr dat pro určenou kameru má být zachován.

Vykonávání programu

Provedení `CamStartLoadJob` pouze přikáže načítání a potom pokračuje přímo s další instrukcí bez čekání, až načítání skončí. Jestliže je použit volitelný argument `\KeepTargets`, starý sběr dat pro určenou kameru není odstraněn. Výchozím chováním je odstranění (vyčištění) starého sběru dat.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.22 CamStartLoadJob - Začít načítat kamerovou úlohu do kamery

Integrated Vision
Pokračování

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_CAM_BUSY	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.

Omezení

Je možné pouze provést CamStartLoadJob, když je kamera nastavena v programovém režimu. Použijte instrukci CamSetProgramMode pro nastavení kamery do programového režimu.

Když se provádí pokračující načítání práce, není možné přistupovat k určené kameře s žádnou jinou instrukcí nebo funkcí. Následující instrukce nebo funkce kamery musí být instrukce CamWaitLoadJob.

Aby bylo možné načíst práci, soubor práce musí být uložen na flash disku kamery.

Syntaxe

```
CamStartLoadJob
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ Name ':='] <expression (IN) of string >
  [ '\'KeepTargets ]';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.23 CamWaitLoadJob – Čekat na načtení úlohy pro kameru

Použití

`CamWaitLoadJob` *(Camera Wait Load Job)* bude čekat, až bude načítání práce do kamery připraveno.

Základní příklady

Následující příklad názorně ukazuje instrukci `CamWaitLoadJob`.

Příklad 1

```
...
CamStartLoadJob mycamera, "myjob.job";
MoveL p1, v1000, fine, tool2;
CamWaitLoadJob mycamera;
CamSetRunMode mycamera;
CamReqImage mycamera;
...
```

Nejdříve je spuštěno načítání práce do kamery, a zatímco načítání probíhá, je proveden pohyb do pozice p1. Když je pohyb připraven a načítání je u konce, je získán obraz.

Argumenty

```
CamWaitLoadJob Camera
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_CAM_COM_TIMEOUT`	Chyba komunikace s kamerou. Kamera je pravděpodobně odpojena.

Omezení

Je možné pouze provést `CamWaitLoadJob`, když je kamera nastavena v programovém režimu. Použijte instrukci `CamSetProgramMode` pro nastavení kamery do programového režimu.

Když se provádí pokračující načítání práce, není možné přistupovat k určené kameře s žádnou jinou instrukcí nebo funkcí. Následující instrukce nebo funkce kamery musí být instrukce `CamWaitLoadJob`.

Syntaxe

```
CamWaitLoadJob
  [ Camera ':=' ] < variable (VAR) of cameradev > ';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.24 CancelLoad - Zrušit načítání modulu

Použití

`CancelLoad` se může používat ke zrušení operace načítání generované od instrukce `StartLoad`.

`CancelLoad` se může používat pouze mezi instrukcí `StartLoad` a `WaitLoad`.

Základní příklady

Následující příklad názorně ukazuje instrukci `CancelLoad`:

Viz také *Další příklady na str 64*.

Example1

```
CancelLoad load1;
```

Činnost načítání `load1` je zrušena.

Argumenty

```
CancelLoad LoadNo
```

`LoadNo`

Datový typ: `loadsession`

Reference k činnosti načítání, vytvořená instrukcí `StartLoad`.

Další příklady

Více příkladů jak používat instrukci `CancelLoad` je názorně uvedeno dole.

Příklad 1

```
VAR loadsession load1;

StartLoad "HOME:"\File:="PART_B.MOD",load1;
...
IF ...
  CancelLoad load1;
  StartLoad "HOME:"\File:="PART_C.MOD",load1;
ENDIF
...
WaitLoad load1;
```

Instrukce `CancelLoad` zruší probíhající načítání modulu `PART_B.MOD` a místo toho umožní načtení `PART_C.MOD`.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_LOADNO_NOUSE`	Proměnná určená v argumentu `LoadNo` se nepoužívá, což znamená, že neprobíhá žádná činnost načítání.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

`CancelLoad` se může používat pouze v sekvenci, po které je instrukce `StartLoad` připravena, a před tím, než je instrukce `WaitLoad` spuštěna.

Syntaxe

```
CancelLoad
  [ LoadNo ':=' ] < variable (VAR) of loadsession >';'
```

Související informace

Pro informace o	Viz
Načíst programový modul během provádění	*StartLoad - Načíst programový modul během vykonávání na str 703*
Připojit načtený modul k úloze	*WaitLoad - Připojit načtený modul k úloze na str 947*
Načíst akci	*loadsession - Načtení programu na str 1530*
Načíst programový modul	*Load - Načíst programový modul během provádění na str 328*
Zrušit načtení programového modulu	*UnLoad - Stáhnout programový modul během provádění na str 905*
Zkontrolovat reference programu	*CheckProgRef - Zkontrolovat reference programu na str 103*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.25 CapAPTrSetup - Nastavení At-Point-Tracker

Použití

`CapAPTrSetup` (*Setup an At-Point-Tracker*) se používá pro nastavení typu senzoru At-Point-Tracker, například, *WeldGuide* nebo AWC.

Rozhraní senzoru komunikuje maximálně s jedním senzorem přes sériové kanály pomocí transportního protokolu RTP1.

Základní příklad

SIO.cfg:

```
COM_PHY_CHANNEL:
-name sio1:" -Connector "COM1"
COM_TRP:
-Name "swg:" -Type "RTP1" -PhyChannel "sio1"
```

Kód RAPID:

```
! Define variable numbers
CONST num SensorOn := 6;
CONST num XCoord := 8;
CONST num YCoord := 9;
CONST num ZCoord := 10;
VAR pos SensorPos;

! Setup a Weldguide
CapAPTrSetup "swg:", do_left, 80, do_right, 80;
```

Argumenty

```
CapAPTrSetup device DoLeft LevelLeft DoRight LevelRight [\LogFile]
     [\LogSize]
```

device

Datový typ: `string`

Jméno I/O zařízení konfigurované v sio.cfg pro použitý senzor.

DoLeft

Datový typ: `signaldo`

Digitální výstupní signál pro weave synchronizaci na levém weave cyklu.

LevelLeft

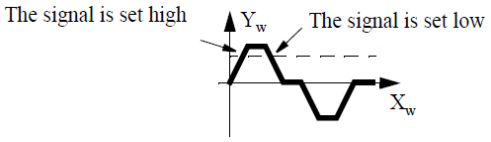
Datový typ: `num`

Koordinační pozice na levé straně weaving obrazce. Určená hodnota je procentní částí šířky vlevo od weaving středu. Když je weaving prováděn za tímto bodem, digitální výstupní signál se automaticky nastaví vysoko (za předpokladu, že signál je definován).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Tento typ koordinace se může používat pro sledování spoje pomocí Through-the-Arc Tracker.



xx1200000178

DoRight

Datový typ: `signaldo`

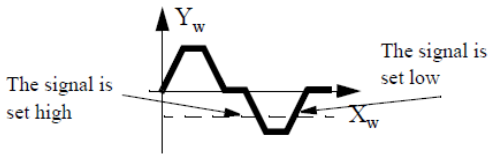
Digitální výstupní signál pro weave synchronizaci na pravém weave cyklu.

LevelRight

Datový typ: `num`

Koordinační pozice na pravé straně weaving obrazce. Určená hodnota je procentní částí šířky vpravo od weaving středu. Když je weaving prováděn za tímto bodem, digitální výstupní signál se automaticky nastaví vysoko (za předpokladu, že signál je definován).

Tento typ koordinace se může používat pro sledování spoje pomocí Through-the-Arc Tracker.



xx1200000179

[\LogFile]

Datový typ: `string`

Jméno logovacího souboru tracklog.

[\LogSize]

Datový typ: `num`

Velikost kruhového bufferu tracklogu, tj. počet měření senzoru, která mohou být uložena do zásobníku během sledování.

Výchozí hodnota: 1000.

Syntaxe

```
CapAPTrSetup
  [device ':='] < expression (IN) of string> ','
  [DoLeft ':='] < expression (IN) of signaldo > ','
  [LevelLeft ':='] < expression (IN) of num > ','
  [DoRight ':='] < expression (IN) of signaldo > ','
  [LevelRight ':='] < expression (IN) of num >
  ['\'LogFile ':='] < expression (IN) of string >
  ['\'LogSize ':='] < expression (IN) of num > ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.25 CapAPTrSetup - Nastavení At-Point-Tracker

Continuous Application Platform (CAP)

Pokračování

Související informace

	Popsáno v:
Continuous Application Platform	*Application manual - Continuous Application Platform*
Sensor Interface	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.26 CapC - Pohybová instrukce cirkulárního CAP

Použití

`CapC` se používá pro posun středního bodu nástroje (TCP) podél kruhové dráhy na dané místo určení a současně ke kontrole probíhajícího procesu. Dále je možné připojit až osm událostí k `CapC`. Události jsou definovány pomocí instrukcí `TriggRampAO`, `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, nebo `TriggSpeed`.

Základní příklady

Příklad 1

Kruhové pohyby s `CapC`.

```
CapC cirp, p1, v100, cdata, weavestart, weave, fine, gun1;
```

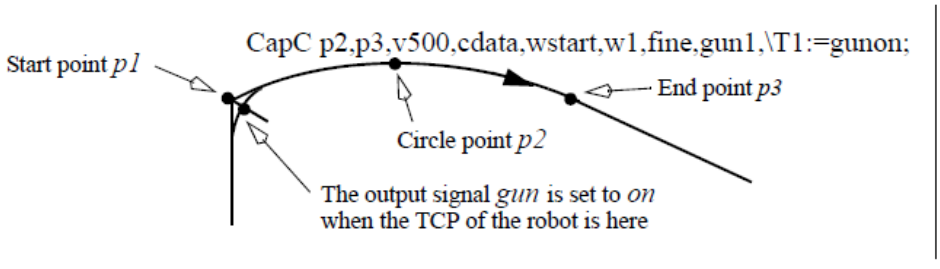
TCP nástroje, `gun1`, se posune kruhově k jemnému bodu `p1` rychlostí definovanou v `cdata`.

Příklad 2

Kruhový pohyb s uživatelskou událostí a CAP událostí.

```
VAR intnum start_intno;
...
PROC main()
  VAR triggdata gunon;

  IDelete start_intno;
  CONNECT start_intno WITH start_trap;
  ICap start_intno, CAP_START;
  TriggIO gunon, 0 \Start \DOp:=gun, on;

  MoveJ p1, v500, z50, gun1;
  CapC p2,p3,v500,cdata,wstart,w1,fine,gun1,\T1:=gunon;
ENDPROC

TRAP start_trap
  ! This routine will be executed when the event CAP_START is
       reported
ENDTRAP
```

Digitální výstupní signál gun je nastaven, když TCP robotu přechází přes midpoint rohové dráhy bodu `p1`. Trap rutina start_trap se provede, když se spouští CAP proces.



xx1200000174

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.26 CapC - Pohybová instrukce cirkulárního CAP

Continuous Application Platform (CAP)
Pokračování

Argumenty

```
CapC Cirpoint ToPoint [\Id] Speed Cdata [\MoveStartTimer] Weavestart
     Weave Zone [\Inpos] Tool [\WObj] [\Track] | [\Corr] [\Time]
     [\T1] [\T2] [\T3] [\T4] [\T5] [\T6] [\T7] [\T8] [\TLoad]
```

`Cirpoint`

Datový typ: robtarget

Kruhový bod robotu. Kruhový bod je bod na kruhu mezi start bodem a bodem určení. Aby bylo dosaženo největší přesnosti, měl by být umístěn asi na polovině vzdálenosti mezi body startu a určení. Jestliže je umístěn blízko bodu startu nebo blízko koncového bodu, robot může dostat varování. Střední bod je definován jako jmenovitá pozice nebo uložen přímo v instrukci (pokud je označeno * v instrukci).

`ToPoint`

Datový typ: robtarget

Bod určení robotu a doplňkových os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[\Id]`

(Sync identity)

Datový typ: identno

Synchronizační identita pro pohybové instrukce RAPID v systému MultiMove v synchronizačním režimu.

`Speed`

Datový typ: speeddata

Rychlostní data, která se vztahují k pohybům bez aktivního procesu CAP. Rychlostní data definují rychlost středního bodu nástroje, přídavných os a reorientace nástroje. Jestliže proces CAP aktivní (neblokovaný), potom argument Cdata definuje rychlost TCP.

`Cdata`

(Data procesu CAP)

Datový typ: capdata

Data procesu CAP, viz *capdata - CAP data na str 1450*, kde je podrobný popis.

`[\Movestart_timer]`

(Čas v sekundách)

Datový typ: num

Horní limit pro časový rozdíl mezi příkazem pro start procesu a skutečným startem TCP pohybu robotu v systému MultiMove v synchronizovaném režimu.

`Weavestart`

(Weavestart Data)

Datový typ: weavestartdata

Spouštěcí data weave pro proces CAP, viz *weavestartdata - spouštěcích data weave na str 1632*, kde najdete podrobný popis.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Weave`

(Weave Data)

Datový typ: capweavedata

Data weaving pro proces CAP, viz *capweavedata - Weavedata pro CAP na str 1463*, kde najdete podrobný popis.

`Zone`

Datový typ: zonedata

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`[\Inpos]`

(In position)

Datový typ: stoppointdata

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

`Tool`

Datový typ: tooldata

Používaný nástroj při pohybu robotu. Střední bod nástroje (TCP) je bod, který se pohybuje do určené pozice.

`[\WObj]`

Datový typ: wobjdata

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnicovému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované dodatečné osy, tento argument musí být stanoven pro lineární pohyb ve vztahu k pracovnímu objektu, který bude proveden.

`[\Track]`

(Track Sensor Data) (Sledovat data senzoru)

Datový typ: captrackdata

Tato datová struktura obsahuje data potřebná k použití senzorům při generování korekce dráhy společně s `CapC`, viz *captrackdata - traťová data CAP na str 1460*. Tento argument není povolen společně s argumentem `\Corr`.

`[\Corr]`

(Použít generátor korekce)

Datový typ: switch

Tento argument přikazuje `CapC` načíst korekce dráhy od generátoru korekcí, viz *CorrCon - Připojuje ke generátoru korekcí na str 140*. Tento argument není povolen společně s argumentem `\Track`.

`[\Time]`

Datový typ: num

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.26 CapC - Pohybová instrukce cirkulárního CAP
Continuous Application Platform (CAP)
Pokračování

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se pohybuje robot a dodatkové osy. Je potom vyměněn za odpovídající rychlostní data.

`[\T1] [\T2] [\T3] [\T4] [\T5] [\T6] [\T7] [\T8]`
(Trigg x)

Datový typ: triggdata

Proměnné, které odkazují ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggRampAO`, `TriggIO`, `TriggEquip`, nebo `TriggInt`.

`[\TLoad]`

Datový typ: loaddata

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom není uvažován `loaddata` v aktuálním `tooldata`.

Když je argument `\TLoad` nastaven na `load0`, potom není argument uvažován a `loaddata` v aktuálním `tooldata` je použit místo něj. Kompletní popis argumentu `TLoad`, viz `MoveL`, *MoveL - Pohybuje robotem lineárně na str 411*.

Řešení chyb

Existuje několik různých typů chyb, které může řešit obslužný program pro chyby pro instrukce `CapC`/`CapL`:

- chyby supervize
- chyby podle senzoru
- chyby specifické pro systém MultiMove
- chyby zděděné po funkčnosti `TriggX`
- další chyby CAP

Jestliže jeden ze signálů, na který by se mělo dohlížet, nemá správnou hodnotu nebo jestliže mění hodnotu během dohlížení, je nastavena systémová proměnná `ERRNO`.

Jestliže není možné načíst žádné hodnoty ze senzoru trati, je nastavena systémová proměnná `ERRNO`.

U systému MultiMove běžícího v synchro režimu musí obslužný program pro chyby řešit dvě jiné chyby. Jedna se používá pro hlášení, že některá jiná aplikace zjistila odstranitelnou chybu. To umožní řešení odstranitelné chyby v synchronizovaných úlohách RAPID. Druhá chyba, `CAP_MOV_WATCHDOG`, je hlášena, když čas mezi příkazem ke startu procesu a skutečným startem TCP pohybu robotů v systému MultiMove v synchro režimu vyprší. Použitý čas je stanoven ve volitelném parametru `Movestart_timer` v instrukci `CapC`.

Jestliže je zjištěno něco abnormálního, provádění programu se zastaví. Jestliže, nicméně, je naprogramován obslužný program pro řešení chyb, chyby definované dole mohou být napraveny bez zastavení produkce. Nicméně, doporučení je takové, že některé chyby (chyby s `CAP_XX`) by neměly být prezentovány koncovému uživateli. Zmapujte tyto chyby jako chyby podle aplikace. U chyb supervize může

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

být použita instrukce `CapGetFailSigs`, aby bylo vidět, který konkrétní signál selhal.

Chyby supervize

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`CAP_PRE_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `PRE`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `pre_cond`).
`CAP_PRESTART_ERR`	Tato chyba se objeví, když existuje chyba během supervize fáze `PRE`.
`CAP_END_PRE_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `END_PRE`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `start_cond`).
`CAP_START_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `START`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `start_cond`).
`CAP_MAIN_ERR`	Tato chyba se objeví, když existuje chyba během supervize hlavní fáze.
`CAP_ENDMAIN_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `END_MAIN`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `end_main_cond`).
`CAP_START_POST1_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `START_POST1`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `end_main_cond`).
`CAP_POST1_ERR`	Tato chyba se objeví, když existuje chyba během supervize fáze `POST1`.
`CAP_POST1END_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `END_POST1`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `end_main_cond`).
`CAP_START_POST2_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `START_POST1`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `end_main_cond`).
`CAP_POST2_ERR`	Tato chyba se objeví, když existuje chyba během supervize fáze `POST2`.
`CAP_POST2END_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `END_POST2`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `end_main_cond`). Jestliže je supervize provedena na dvou odlišných signálech ve stejné fázi a oba selžou, první z nich, který je potom nastaven s `SetupSuperv`, je signál, který generuje chybu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.26 CapC - Pohybová instrukce cirkulárního CAP

Continuous Application Platform (CAP)

Pokračování

Chyby podle senzoru

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

CAP_TRACK_ERR	Chyba sledování se objeví, když jsou načítána data ze senzoru a po čase nejsou žádná platná data přijata. Jedním z důvodů může být to, že senzor nemůže indikovat spoj.
CAP_TRACKSTA_ERR	Chyba startu sledování vznikne, když nebyla načtena žádná platná data od laserového sledovacího senzoru.
CAP_TRACKCOR_ERR	Chyba korekce trati vznikne, když něco neprobíhá správně ve výpočtu ofsetu.
CAP_TRACKCOM_ERR	Komunikace mezi řadičem robotu a senzorovým vybavením je přerušena.
CAP_TRACKPFR_ERR	Není možné pokračovat ve sledování, jestliže k selhání napájení došlo během sledování.
CAP_SEN_NO_MEAS	Řadič nedostal platné měření od senzoru.
CAP_SEN_NOREADY	Senzor není dosud připraven.
CAP_SEN_GENERRO	Došlo k chybě obecného senzoru.
CAP_SEN_BUSY	Senzor je zaměstnán a nemůže odpovědět na požadavek.
CAP_SEN_UNKNOWN	Příkaz odeslaný k senzoru je pro senzor neznámý.
CAP_SEN_ILLEGAL	Proměnná nebo číslo bloku odeslané k senzoru jsou zakázané.
CAP_SEN_EXALARM	Externí alarm vznikl v senzoru.
CAP_SEN_CAALARM	Alarm kamery vznikl v senzoru.
CAP_SEN_TEMP	Teplota senzoru je mimo rozsah.
CAP_SEN_VALUE	Hodnota odeslaná k senzoru je mimo rozsah.
CAP_SEN_CAMCHECK	Selhala kontrola kamery.
CAP_SEN_TIMEOUT	Senzor nereagoval během určeného času.

Chyby možné v systémech MultiMove

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_PATH_STOP	Při používání synchro pohybu je tato chyba hlášena, když aplikace kontrolující jednu mechanickou jednotku zjistí řešitelnou chybu a upozorní ostatní aplikace, že něco není v pořádku. Jestliže je tento chybový kód přijat od instrukce CapC, chyba je reakcí na jinou chybu. Všechny úlohy používající pohybové instrukce v synchro režimu v systému MultiMove by měly mít tuto hodnotu ERRNO definovanou v obslužném programu pro řešení chyb.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Chyby zděděné po TriggX

Instrukce CapC je založena na instrukci TriggC. Jako důsledek můžete dostat a řešit chyby ERR_AO_LIM a ERR_DIPLAG_LIM, jako v TriggC.

Systémová proměnná ERRNO bude nastavena na:

ERR_AO_LIM	Jestliže výsledky naprogramovaného argumentu ScaleValue/SetValue pro stanovený analogový výstupní signál AOp/AOutput v některé z připojených instrukcí TriggSpeed/TriggRampAO jsou mimo limit pro analogový signál společně s naprogramovaným Speed v této instrukci. Systémová proměnná ERRNO je nastavena na ERR_AO_LIM.
ERR_DIPLAG_LIM	Jestliže naprogramovaný argument DipLag v některé z připojených instrukcí TriggSpeed je příliš velký ve vztahu k použitému systémovému parametru *Event Preset Time*, systémová proměnná ERRNO je nastavena na ERR_DIPLAG_LIM.

Jiné chyby CAP

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

CAP_NOPROC_END	Tyto chyba vznikne, když se použije instrukce CapNoProcess pro běh určité distance bez aplikačního procesu a je dosaženo konce této distance. Není to skutečná chyba, ale používá mechanismu zotavení po chybě.
CAP_MOV_WATCHDOG	Tato chyba vznikne, když je stanoven přepínač \Movestart_timer a čas mezi startem procesu (MAIN_STARTED) a startem pohybu robotu překročí čas stanovený se spínačem.

Vykonávání programu

Viz *MoveL - Pohybuje robotem lineárně na str 411*, kde jsou informace o lineárním pohybu.

Viz *TriggL - Lineární pohyby robotu s událostmi na str 839*, kde najdete informace o lineárním pohybu se spouštěcími událostmi.

Proces CAP

Během plynulého provádění v Auto režimu a Ručním režimu běží proces CAP, pokud není blokován. To znamená, že veškerá data řídící process CAL (tj. Cdata, Weavestart, Weave a Movestart_timer), se používají. V těchto režimech jsou prováděny spouštěcí aktivity CAP, viz *ICap - připojit události CAP k trap rutinám na str 238*.

Ve všech ostatních režimech provádění proces CAP neběží a instrukce CapC se chová jako instrukce MoveC.

Spouštěcí podmínky [\T1] až [\T8]

Jelikož spouštěcí podmínky jsou splněny, když je robot umístěn blíž a blíž ke koncovému bodu, definované spouštěcí aktivity jsou provedeny. Spouštěcí podmínky jsou splněny buď v určité vzdálenosti před koncovým bodem instrukce nebo v určité vzdálenosti po bodu startu instrukce nebo v určitém časovém bodu (omezeno na krátký čas) před koncovým bodem instrukce.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.26 CapC - Pohybová instrukce cirkulárního CAP

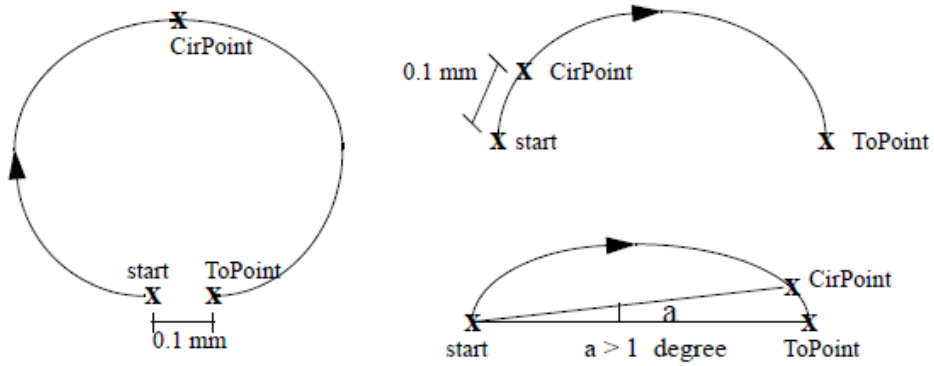
Continuous Application Platform (CAP)

Pokračování

Během krokového provádění dopředu jsou I/O aktivity provedeny, ale rutiny přerušení neběží. Během krokového provádění dozadu nejsou prováděny vůbec žádné spouštěcí aktivity.

Omezení

Existují omezení v tom, jak může být umístěno *CirPoint* a *ToPoint*, jak ukazuje obrázek dole.



xx1200000175

- Min. vzdálenost mezi startem a ToPoint je 0,1 mm.
- Min. vzdálenost mezi startem a CirPoint je 0,1 mm.
- Min. úhel mezi CirPoint a ToPoint od bodu startu je 1 stupeň.

Přesnost může být špatná blízko limitů, například, jestliže bod startu a ToPoint na kružnici jsou blízko sebe, porucha způsobená sklonem kruhu může být mnohem větší než přesnost, se kterou byly body programovány.

Změna režimu provádění od dopředu na dozadu nebo opačně, zatímco robot je zastaven na kruhové dráze, není přípustná a výsledkem bude chybová zpráva.

Instrukce `CapC` (nebo jiná instrukce obsahující kruhový pohyb) by nikdy neměla být spouštěna od začátku, s TCP mezi bodem kruhu a koncovým bodem. Jinak robot nevezme naprogramovanou dráhu (polohování kolem kruhové dráhy v jiném směru v porovnání s naprogramovanou).

Zajistěte, aby robot mohl dosáhnout kruhového bodu během provádění programu a podle potřeby rozdělte segment kruhu.

Jestliže se aktuální bod startu odchyluje od obvyklého, takže celková délka polohování instrukce `CapC` je kratší než obvykle, může se stát, že několik nebo všechny spouštěcí podmínky jsou okamžitě splněny na stejné pozici. V takových případech bude sekvence, ve které jsou spouštěcí aktivity prováděny, nedefinována. Programová logika v uživatelském programu nesmí být založena na normální sekvenci spouštěcích aktivit pro „nekompletní pohyb“.

Syntaxe

```
CapC
  [CirPoint ':='] < expression (IN) of robtarget >
  [ToPoint ':='] < expression (IN) of robtarget >
  ['\' Id ':=' < expression (IN) of identno > ] ','
  [Speed ':='] < expression (IN) of speeddata >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
[Cdata ':='] < persistent (PERS) of capdata >
['\' Movestart_timer ':=' < expression (IN) of num > ] ','
[Weavestart ':='] <persistent (PERS) of weavestartdata >
[Weave ':='] < persistent (PERS) of capweavedata >
[Zone ':='] < expression (IN) of zonedata >
['\' Inpos ':=' < expression (IN) of stoppointdata >] ','
[Tool ':='] < persistent (PERS) of tooldata >
['\' WObj ':=' < persistent (PERS) of wobjdata > ]
['\' Track ':=' < persistent (PERS) of captrackdata > ]
|[ '\' Corr]
['\' Time ':=' < expression (IN) of num > ]
['\' T1 ':=' < variable (VAR) of triggdata > ]
['\' T2 ':=' < variable (VAR) of triggdata > ]
['\' T3 ':=' < variable (VAR) of triggdata > ]
['\' T4 ':=' < variable (VAR) of triggdata > ]
['\' T5 ':=' < variable (VAR) of triggdata > ]
['\' T6 ':=' < variable (VAR) of triggdata > ]
['\' T7 ':=' < variable (VAR) of triggdata > ]
['\' T8 ':=' < variable (VAR) of triggdata > ]
['\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Kruhový pohyb	*MoveC - Pohybuje robotem kruhově na str 358*
Kruhový pohyb se spouštěči	*TriggC - Kruhový pohyb robotu s událostmi na str 796*
Definice dat CAP	*capdata - CAP data na str 1450*
Definice spouštěcích dat weave	*weavestartdata - spouštěcích data weave na str 1632*
Definice dat weave	*capweavedata - Weavedata pro CAP na str 1463*
Definice dat track	*captrackdata - traťová data CAP na str 1460*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.27 CapCondSetDO - Nastavit digitální výstupní signál na TCP stop

Použití

`CapCondSetDO` se používá k definování digitálního výstupního signálu a jeho hodnoty, která bude nastavena, až TCP robotu, který provádí CAP, zastaví pohyb během instrukce CAP (`CapL` or `CapC`) před dokončením sekvence CAP.

Existující definice takových signálů je vyčištěna instrukcí CAP `InitSuperv`.

Základní příklad

```
CapCondSetDO do15, 1;
```

Signál do15 je nastaven na 1, když se TCP zastaví.

```
CapCondSetDO weld, off;
```

Signál weld (svar) je nastaven na `off`, když se TCP zastaví.

Argumenty

```
CapCondSetDO Signal Value
```

Signal

Datový typ: `signaldo`

Jméno signálu, který bude změněn.

Value

Datový typ: `dionum`

Požadovaná hodnota signálu 0 nebo 1.

Stanovená Value	Nastavit digitální výstup na
0	0
Každá hodnota kromě 0	1

Omezení

Konečná hodnota signálu závisí na konfiguraci signálu. Jestliže je signál invertován v systémových parametrech, hodnota fyzického kanálu je opačná.

Může být nastaveno max. 10 signálů na jednu úlohu RAPID.

Syntaxe

```
CapCondSetDO
  [Signal ':='] < variable (VAR) of signaldo > ','
  [Value ':='] < expression (IN) of dionum > ';'
```

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instrukce `InitSuperv`	*InitSuperv - Resetovat veškerý dohled pro CAP na str 271*
Instrukce `SetupSuperv`	*SetupSuperv - Nastavit podmínky pro dohled signálu v CAP na str 639*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Instrukce `RemoveSuperv`	*RemoveSuperv - Odstranit podmínku pro jeden signál na str 538*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.28 CapEquiDist - Generovat ekvidistantní událost

Použití

`CapEquiDist` se používá ke sdělení CAP generovat ekvidistantní událost RAPID (EQUIDIST) na dráze CAP. První událost je generována na bodu startu první instrukce CAP v sekvenci CAP instrukcí. Od RAPID je možné přihlásit tuto událost pomocí `ICap`.

Základní příklad

```
VAR intnum intno_equi;

PROC main()
  ......
  IDelete intno_equi;
  Connect intno_equi equi_trp;
  ICap intno_equi, EQUIDIST
  ......
  CapEquiDist\Distance:=5.0;
  MoveL p60, v1000, fine, tWeldGun;
  CapL p_fig3_l_1, v500, cd, wsd, cwd, z10, tWeldGun;
  CapL p_fig3_l_2, v500, cd, wsd, cwd, fine, tWeldGun;
  ......
  CapEquiDist\Reset;
  MoveL p70, v1000, fine, tWeldGun;
  CapL p_fig3_l_3, v500, cd, wsd, cwd, fine, tWeldGun;
  ......

  ERROR
    Retry;
ENDPROC

TRAP equi_trp
  ! do whatever you want, but it must not take too long time
ENDTRAP
```

V tomto příkladu bude generována událost `EQUIDIST` na první dráze CAP. Bude odesílána každých 5 mm na dráze přes několik CAP instrukcí se zónami.

Argumenty

```
CapEquiDist [\Distance] [\Reset]
```

[\Distance]

Vzdálenost v mm

Datový typ: `num`

Data s tímto volitelným argumentem definují vzdálenost v mm mezi dvěma po sobě jdoucími ekvidistantními událostmi.

[\Reset]

Resetovat generování události

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Datový typ: `switch`

Jestliže je přítomen přepínač, generování události je resetováno, to znamená, že ekvidistantní událost nebude nadále generována na dráze `CapL`/`CapC`. Tento přepínač má přednost před přepínačem `\Distance`.

Omezení

Jestliže dráha CAP je dlouhá ve srovnání se vzdáleností události, systém může spotřebovat zdroje událostí a objeví se chybová zpráva **50368 Příliš krátká vzdálenost mezi ekvidistantnímu událostmi**.

Syntaxe

```
CapEquiDist
  ['\' Distance ':=' < expression (IN) of num >]
  ['\' Reset] ';'
```

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.29 CapL - Instrukce pro lineární pohyb CAP

Použití

`CapL` se používá pro posun středního bodu nástroje (TCP) lineárně na dané místo určení a současně ke kontrole probíhajícího procesu. Dále je možné připojit až osm událostí k `CapL`. Události jsou definovány pomocí instrukcí `TriggRampAO`, `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, nebo `TriggSpeed`.

Základní příklady

Example1

Lineární pohyby s `CapL`.

```
CapL p1, v100, cdata, weavestart, weave, z50, gun1;
```

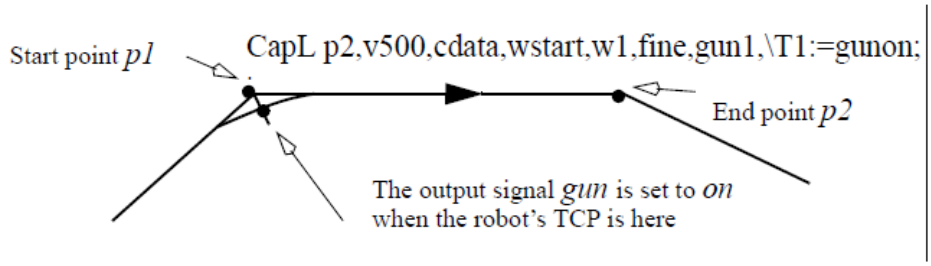
TCP nástroje, gun1, se posune lineárně k pozici p1, rychlostí definovanou v `cdata`, a zónových datech z50.

Příklad 2

Kruhový pohyb s uživatelskou událostí a CAP událostí.

```
VAR intnum start_intno;
...
PROC main()
  VAR triggdata gunon;
  IDelete start_intno;
  CONNECT start_intno WITH start_trap;
  ICap start_intno, CAP_START;
  TriggIO gunon, 0 \Start \DOp:=gun, on;
  MoveJ p1, v500, z50, gun1;
  CapL p2, v500, cdata, wstart, w1, fine, gun1 \T1:=gunon;
ENDPROC

TRAP start_trap
  !This routine is executed when event CAP_START arrives
ENDTRAP
```

Digitální výstupní signál gun je nastaven, když TCP robotu přechází přes midpoint rohové dráhy bodu p1. Rutina trap `start_trap` se provede, když startuje proces CAP.



xx1200000173

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Argumenty

```
CapLToPoint [\Id] Speed Cdata [\MoveStartTimer] Weavestart Weave
    Zone [\Inpos] Tool [\WObj] [\Track] | [\Corr] [\Time] [\T1]
    [\T2] [\T3] [\T4] [\T5] [\T6] [\T7] [\T8] [\TLoad]
```

`ToPoint`

Datový typ: robtarget

Bod určení robotu a doplňkových os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \Id ]`

(Sync identity)

Datový typ: identno

Synchronizační identita pro pohybové instrukce RAPID v systému MultiMove v synchronizačním režimu.

`Speed`

Datový typ: speeddata

Rychlostní data, která se vztahují k pohybům bez aktivního procesu CAP. Rychlostní data definují rychlost středního bodu nástroje, přídavných os a reorientace nástroje. Jestliže proces CAP aktivní (neblokovaný), potom argument Cdata definuje rychlost TCP.

`Cdata`

(Data procesu CAP)

Datový typ: capdata

Data procesu CAP, viz *capdata - CAP data na str 1450*, kde je podrobný popis.

`[ \Movestart_timer ]`

(Čas v sekundách)

Datový typ: num

Horní limit pro časový rozdíl mezi příkazem pro start procesu a skutečným startem TCP pohybu robotu v systému MultiMove v synchronizovaném režimu.

`Weavestart`

(Weavestart Data)

Datový typ: weavestartdata

Spouštěcí data weave pro proces CAP, viz *weavestartdata - spouštěcích data weave na str 1632*, kde najdete podrobný popis.

`Weave`

(Weave Data)

Datový typ: capweavedata

Data weaving pro proces CAP, viz *capweavedata - Weavedata pro CAP na str 1463*, kde najdete podrobný popis.

`Zone`

Datový typ: zonedata

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.29 CapL - Instrukce pro lineární pohyb CAP

Continuous Application Platform (CAP)
Pokračování

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`[\Inpos ]`

(In position)

Datový typ: stoppointdata

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

`Tool`

Datový typ: tooldata

Používaný nástroj při pohybu robotu. Střední bod nástroje (TCP) je bod, který se pohybuje do určené pozice.

`[ \WObj ]`

Datový typ: wobjdata

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnicovému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované dodatečné osy, tento argument musí být stanoven pro lineární pohyb ve vztahu k pracovnímu objektu, který bude proveden.

`[ \Track ]`

(Track Sensor Data) (Sledovat data senzoru)

Datový typ: captrackdata

Tato datová struktura obsahuje data potřebná k použití senzorům při generování korekce dráhy společně s `CapL`, viz *captrackdata - traťová data CAP na str 1460*. Tento argument není povolen společně s argumentem `\Corr`.

`[ \Corr ]`

(Použít generátor korekce)

Datový typ: switch

Tento argument přikazuje `CapL` načíst korekce dráhy od generátoru korekcí, viz *CorrCon - Připojuje ke generátoru korekcí na str 140*. Tento argument není povolen společně s argumentem `\Track`.

`[ \Time ]`

Datový typ: num

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se pohybuje robot a dodatkové osy. Je potom vyměněn za odpovídající rychlostní data.

`[ \T1 ]` **až** `[ \T8 ]`

(Trigg x)

Datový typ: triggdata

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Proměnné, které odkazují ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggRampAO`, `TriggIO`, `TriggEquip`, nebo `TriggInt`.

`[\TLoad]`

Datový typ: loaddata

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom není uvažován `loaddata` v aktuálním `tooldata`.

Když je argument `\TLoad` nastaven na `load0`, potom není argument uvažován a `loaddata` v aktuálním `tooldata` je použit místo něj. Kompletní popis argumentu `TLoad`, viz `MoveL`, *MoveL - Pohybuje robotem lineárně na str 411*.

Řešení chyb

Existuje několik různých typů chyb, které může řešit obslužný program pro chyby pro instrukce `CapC`/`CapL`:

- chyby supervize
- chyby podle senzoru
- chyby specifické pro systém MultiMove
- chyby zděděné po funkčnosti `TriggX`
- další chyby CAP

Jestliže jeden ze signálů, na který by se mělo dohlížet, nemá správnou hodnotu nebo jestliže mění hodnotu během dohlížení, je nastavena systémová proměnná `ERRNO`.

Jestliže není možné načíst žádné hodnoty ze senzoru trati, je nastavena systémová proměnná `ERRNO`.

U systému MultiMove běžícího v synchro režimu musí obslužný program pro chyby řešit dvě jiné chyby. Jedna se používá pro hlášení, že některá jiná aplikace zjistila odstranitelnou chybu. To umožní řešení odstranitelné chyby v synchronizovaných úlohách RAPID. Druhá chyba, `CAP_MOV_WATCHDOG`, je hlášena, když čas mezi příkazem ke startu procesu a skutečným startem TCP pohybu robotů v systému MultiMove v synchro režimu vyprší. Použitý čas je stanoven ve volitelném parametru `Movestart_timer` v instrukci `CapL`.

Jestliže je zjištěno něco abnormálního, provádění programu se zastaví. Jestliže, nicméně, je naprogramován obslužný program pro řešení chyb, chyby definované dole mohou být napraveny bez zastavení produkce. Nicméně, doporučení je takové, že některé chyby (chyby s `CAP_XX`) by neměly být prezentovány koncovému uživateli. Zmapujte tyto chyby jako chyby podle aplikace. U chyb supervize může

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

být použita instrukce `CapGetFailSigs`, aby bylo vidět, který konkrétní signál selhal.

Chyby supervize

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`CAP_PRE_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `PRE`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `pre_cond`).
`CAP_PRESTART_ERR`	Tato chyba se objeví, když existuje chyba během supervize fáze `PRE`.
`CAP_END_PRE_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `END_PRE`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `start_cond`).
`CAP_START_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `START`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `start_cond`).
`CAP_MAIN_ERR`	Tato chyba se objeví, když existuje chyba během supervize hlavní fáze.
`CAP_ENDMAIN_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `END_MAIN`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `end_main_cond`).
`CAP_START_POST1_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `START_POST1`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `end_main_cond`).
`CAP_POST1_ERR`	Tato chyba se objeví, když existuje chyba během supervize fáze `POST1`.
`CAP_POST1END_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `END_POST1`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `end_main_cond`).
`CAP_START_POST2_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `START_POST1`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `end_main_cond`).
`CAP_POST2_ERR`	Tato chyba se objeví, když existuje chyba během supervize fáze `POST2`.
`CAP_POST2END_ERR`	Tato chyba se objeví, když je chyba v seznamu supervize `END_POST2`, to znamená, když se nesejdou podmínky v seznamu s určeným časovým rámcem (stanoveným v `end_main_cond`). Jestliže je supervize provedena na dvou odlišných signálech ve stejné fázi a oba selžou, první z nich, který je potom nastaven s `SetupSuperv`, je signál, který generuje chybu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Chyby podle senzoru

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

CAP_TRACK_ERR	Chyba sledování se objeví, když jsou načítána data ze senzoru a po čase nejsou žádná platná data přijata. Jedním z důvodů může být to, že senzor nemůže indikovat spoj.
CAP_TRACKSTA_ERR	Chyba startu sledování vznikne, když nebyla načtena žádná platná data od laserového sledovacího senzoru.
CAP_TRACKCOR_ERR	Chyba korekce trati vznikne, když něco neprobíhá správně ve výpočtu ofsetu.
CAP_TRACKCOM_ERR	Komunikace mezi řadičem robotu a senzorovým vybavením je přerušena.
CAP_TRACKPFR_ERR	Není možné pokračovat ve sledování, jestliže k selhání napájení došlo během sledování.
CAP_SEN_NO_MEAS	Řadič nedostal platné měření od senzoru.
CAP_SEN_NOREADY	Senzor není dosud připraven.
CAP_SEN_GENERRO	Došlo k chybě obecného senzoru.
CAP_SEN_BUSY	Senzor je zaměstnán a nemůže odpovědět na požadavek.
CAP_SEN_UNKNOWN	Příkaz odeslaný k senzoru je pro senzor neznámý.
CAP_SEN_ILLEGAL	Proměnná nebo číslo bloku odeslané k senzoru jsou zakázané.
CAP_SEN_EXALARM	Externí alarm vznikl v senzoru.
CAP_SEN_CAALARM	Alarm kamery vznikl v senzoru.
CAP_SEN_TEMP	Teplota senzoru je mimo rozsah.
CAP_SEN_VALUE	Hodnota odeslaná k senzoru je mimo rozsah.
CAP_SEN_CAMCHECK	Selhala kontrola kamery.
CAP_SEN_TIMEOUT	Senzor nereagoval během určeného času.

Chyby možné v systémech MultiMove

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_PATH_STOP	Při používání synchro pohybu je tato chyba hlášena, když aplikace kontrolující jednu mechanickou jednotku zjistí řešitelnou chybu a upozorní ostatní aplikace, že něco není v pořádku. Jestliže je tento chybový kód přijat od instrukce CapL, chyba je reakcí na jinou chybu. Všechny úlohy používající pohybové instrukce v synchro režimu v systému MultiMove by měly mít tuto hodnotu ERRNO definovanou v obslužném programu pro řešení chyb.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.29 CapL - Instrukce pro lineární pohyb CAP

Continuous Application Platform (CAP)
Pokračování

Chyby zděděné po TriggX

Instrukce CapL je založena na instrukci TriggL. Jako důsledek můžete dostat a řešit chyby ERR_AO_LIM a ERR_DIPLAG_LIM, jako v TriggL.

Systémová proměnná ERRNO bude nastavena na:

ERR_AO_LIM	Jestliže výsledky naprogramovaného argumentu ScaleValue/SetValue pro stanovený analogový výstupní signál AOp/AOutput v některé z připojených instrukcí TriggSpeed/TriggRampAO jsou mimo limit pro analogový signál společně s naprogramovaným Speed v této instrukci. Systémová proměnná ERRNO je nastavena na ERR_AO_LIM.
ERR_DIPLAG_LIM	Jestliže naprogramovaný argument DipLag v některé z připojených instrukcí TriggSpeed je příliš velký ve vztahu k použitému systémovému parametru *Event Preset Time*, systémová proměnná ERRNO je nastavena na ERR_DIPLAG_LIM.

Jiné chyby CAP

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

CAP_NOPROC_END	Tyto chyba vznikne, když se použije instrukce CapNoProcess pro běh určité distance bez aplikačního procesu a je dosaženo konce této distance. Není to skutečná chyba, ale používá mechanismu zotavení po chybě.
CAP_MOV_WATCHDOG	Tato chyba vznikne, když je stanoven přepínač \Movestart_timer a čas mezi startem procesu (MAIN_STARTED) a startem pohybu robotu překročí čas stanovený se spínačem.

Vykonávání programu

Viz *MoveL - Pohybuje robotem lineárně na str 411*, kde jsou informace o lineárním pohybu.

Viz *TriggL - Lineární pohyby robotu s událostmi na str 839*, kde najdete informace o lineárním pohybu se spouštěcími událostmi.

Proces CAP

Během plynulého provádění v Auto režimu a Ručním režimu běží proces CAP, pokud není blokován. To znamená, že veškerá data řídící process CAL (tj. Cdata, Weavestart, Weave a Movestart_timer), se používají. V těchto režimech jsou prováděny spouštěcí aktivity CAP, viz *ICap - připojit události CAP k trap rutinám na str 238*.

Ve všech ostatních režimech provádění proces CAP neběží a instrukce CapL se chová jako instrukce MoveL.

Spouštěcí podmínky [\T1] až [\T8]

Jelikož spouštěcí podmínky jsou splněny, když je robot umístěn blíž a blíž ke koncovému bodu, definované spouštěcí aktivity jsou provedeny. Spouštěcí podmínky jsou splněny buď v určité vzdálenosti před koncovým bodem instrukce nebo v určité vzdálenosti po bodu startu instrukce nebo v určitém časovém bodu (omezeno na krátký čas) před koncovým bodem instrukce.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Během krokového provádění dopředu jsou I/O aktivity provedeny, ale rutiny přerušení neběží. Během krokového provádění dozadu nejsou prováděny vůbec žádné spouštěcí aktivity.

Omezení

Jestliže se aktuální bod startu liší od obvyklého, takže celková polohovací délka instrukce `CapL` je kratší než obvykle (například při spuštění `CapL` s pozicí robotu na koncovém bodě), může se stát, že několik nebo všechny spouštěcí podmínky jsou splněny okamžitě a na stejné pozici. V takových případech bude sekvence, ve které jsou spouštěcí aktivity prováděny, nedefinována. Programová logika v uživatelském programu nesmí být založena na normální sekvenci spouštěcích aktivit pro „nekompletní pohyb“.

Chování CAP procesu může být nedefinováno, jestliže se objeví chyba během instrukcí `CapL` or `CapC` s extrémně krátkými pohyby TCP (< 1 mm).

Syntaxe

```
CapL
  [ToPoint ':='] < expression (IN) of robtarget >
  ['\' Id ':=' < expression (IN) of identno >] ','
  [Speed ':='] < expression (IN) of speeddata > ','
  [Cdata ':='] < persistent (PERS) of capdata >
  ['\' Movestart_timer ':=' < expression (IN) of num >] ','
  [Weavestart ':='] <persistent (PERS) of weavestartdata > ','
  [Weave ':='] < persistent (PERS) of capweavedata > ','
  [Zone ':='] < expression (IN) of zonedata >
  ['\' Inpos ':=' < expression (IN) of stoppointdata >] ','
  [Tool ':='] < persistent (PERS) of tooldata >
  ['\' WObj ':=' < persistent (PERS) of wobjdata >]
  ['\' Track ':=' < persistent (PERS) of captrackdata >]
  |[ '\' Corr]
  ['\' Time ':=' < expression (IN) of num >]
  ['\' T1 ':=' < variable (VAR) of triggdata >]
  ['\' T2 ':=' < variable (VAR) of triggdata >]
  ['\' T3 ':=' < variable (VAR) of triggdata >]
  ['\' T4 ':=' < variable (VAR) of triggdata >]
  ['\' T5 ':=' < variable (VAR) of triggdata >]
  ['\' T6 ':=' < variable (VAR) of triggdata >]
  ['\' T7 ':=' < variable (VAR) of triggdata >]
  ['\' T8 ':=' < variable (VAR) of triggdata >]
  ['\' TLoad':=' < persistent (PERS) of loaddata >] ';'
```

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Lineární pohyb	*MoveL - Pohybuje robotem lineárně na str 411*
Lineární pohyb se spouštěči	*TriggL - Lineární pohyby robotu s událostmi na str 839*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.29 CapL - Instrukce pro lineární pohyb CAP

Continuous Application Platform (CAP)

Pokračování

Pro informace o	Viz
Definice dat CAP	*capdata - CAP data na str 1450*
Definice spouštěcích dat weave	*weavestartdata - spouštěcích data weave na str 1632*
Definice dat weave	*capweavedata - Weavedata pro CAP na str 1463*
Definice dat track	*captrackdata - traťová data CAP na str 1460*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.30 CapLATrSetup - Nastavit Look-Ahead-Tracker

Použití

`CapLATrSetup` (*Set up a Look-Ahead-Tracker*) se používá pro nastavení typu senzoru Look-Ahead-Tracker, například Laser Tracker.

Rozhraní senzoru komunikuje maximálně se dvěma senzory přes sériové kanály pomocí transportního protokolu RTP1. Tyto dva kanály musí být pojmenovány *laser1:* a *swg:*.

Základní příklad

SIO.cfg:

```
COM_TRP:
-Name "SCOUT:" -Type "RTP1"
-Name "digi-ip:" -Type "SOCKDEV" -PhyChannel "LAN1" -RemoteAdress
     "192.168.125.5"
```

Kód RAPID:

```
! Define variable numbers
CONST num SensorOn := 6;
CONST num XCoord := 8;
CONST num YCoord := 9;
CONST num ZCoord := 10;
! Sensor calibration frame
PERS pose calibFrame := [[236.4,0.3,96.3],[1,0,0,0]];
! Trackdata
PERS captrackdata captrack1 := ["digi-ip:", [1,10,1,0,0,0,0,0]];

! Set up a Laser Tracker
CapLATrSetup “digi-ip:”,
     calibFrame\SensorFreq:=20\CorrFilter:=5\MaxBlind:=100\MaxIncCorr:=2;

! Request start of sensor measurements
WriteVar "digi-ip:", SensorOn, 1;

! Track using Cap
CapL p_fig1_l_1, v200, cd_event1, wsd_event, cwd_event, z20,
     tWeldGun\Track:=captrack1;

! Stop sensor
WriteVar "digi-ip:", SensorOn, 0;
```

Argumenty

```
CapLATrSetup device CalibFrame CalibPos [\WarnMaxCorr] [\LogFile]
     [\LogSize] [\SensorFreq] [\IpolServoDelay] [\IpolCorrGain]
     [\ServoSensFactor] [\CorrFilter] [\IpolCorrFilter]
     [\ServoCorrFilter] [\ErrRampIn] [\ErrRampOut] [\CBAngle]
     [\MaxBlind] [\MaxIncCorr] [\CalibFrame2] [\CalibFrame3]
```

device

Datový typ: `string`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.30 CapLATrSetup - Nastavit Look-Ahead-Tracker
Continuous Application Platform (CAP)
Pokračování

Jméno zařízení, jak je definováno v sio.cfg.

calibframe

Datový typ: `pose`

Kalibrační rámec LATR (pozice a orientace vztažená k předdefinovanému nástroji tool0.

CalibPos

Datový typ: `pose`

Kalibrační ofset LATR. Seřízení rámce senzoru, což umisťuje origo rámce korekce dráhy poblíž úrovně rámce nástroje použitého během kalibrace.

[\WarnMaxCorr]

Datový typ: `switch`

Jestliže je tento přepínač přítomen, provádění programu se nepřeruší, když je překročen limit pro max. korekci stanovenou v trackdata. Bude odesláno pouze varování.

[\Logfile]

Datový typ: `string`

Jméno logovacího souboru tracklog.

[\LogSize]

Datový typ: `num`

Velikost kruhového bufferu tracklogu, tj. počet měření senzoru, která mohou být uložena do zásobníku během sledování.

Výchozí: 1000.

[\SensorFreq]

Datový typ: `num`

Definuje vzorkovací kmitočet použitého senzoru (např. M-Spot-90 má vzorkovací kmitočet 5 Hz).

Nejvyšší dostupná hodnota je nezávislá na komunikační lince a její rychlosti. Doporučujeme nepoužívat hodnoty vyšší než 20 Hz.

Výchozí: 5 Hz.

[\IpolServoDelay]

Datový typ: `num`

Definuje vnitřní čas řadiče robotu mezi úlohou ipol a úlohou servo.

Výchozí: 74 ms.

POZNÁMKA

Neměňte výchozí hodnotu!

[\IpolCorrGain]

Datový typ: `num`

Definuje ziskový faktor pro korekci uloženou na ipol.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Výchozí: 0.0.

POZNÁMKA

Neměňte výchozí hodnotu!

[\ServoSensFactor]

Datový typ: num

Definuje počet servo korekcí na jedno načtení senzoru.

Výchozí: 0.

POZNÁMKA

Neměňte výchozí hodnotu!

[\CorrFilter]

Datový typ: num

Definuje filtrování vypočítané korekce pomocí střední hodnoty přes corr hodnoty filtru.

Výchozí: 1.

POZNÁMKA

Neměňte výchozí hodnotu!

[\IpolCorrFilter]

Datový typ: num

Definuje filtrování korekce ipol pomocí střední hodnoty přes hodnoty filtru dráhy.

Výchozí: 1.

POZNÁMKA

Neměňte výchozí hodnotu!

[\ServoCorrFilter]

Datový typ: num

Definuje filtrování servo korekce pomocí střední hodnoty přes hodnoty servo filtru dráhy.

Výchozí: 1.

POZNÁMKA

Neměňte výchozí hodnotu!

[\ErrRampIn]

Datový typ: num

Definuje během kolika čtení senzoru je provedeno ramp in po chybě způsobené čtením senzoru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.30 CapLATrSetup - Nastavit Look-Ahead-Tracker
Continuous Application Platform (CAP)
Pokračování

Výchozí: 1.

[\ErrorRampOut]

Datový typ: num

Definuje během kolika čtení senzoru je provedeno ramp out po chybě způsobené čtením senzoru.

Výchozí: 1.

[\CBAngle]

Datový typ: num

Definuje úhel mezi paprskem 3D senzoru a osou Z senzoru.

Výchozí: 0.0.

[\MaxBlind]

Datový typ: num

Max. vzdálenost, na kterou se TCP smí posunout, jestliže je poslední korekce stále platná.

Na začátku sledování je vzdálenost MaxBlind automaticky zvětšena dopředným pohledem senzoru.

Výchozí hodnota: bez limitu.

[\MaxIncCorr]

Datový typ: num

Max. povolená přírůstková korekce.

Jestliže přírůstková TCP korekce je větší než u \MaxIncCorr a \WarnMaxCorr bylo stanoveno, robot bude pokračovat svoji dráhu, ale aplikovaná přírůstková korekce nepřekročí \MaxIncCorr. Jestliže nebylo stanoveno \WarnMaxCorr, chyba trasy je hlášena a provádění programu se zastaví.

Výchozí: 5 mm.

[\CalibFrame2]

Datový typ: pose

Číslo alternativního kalibračního rámce LATR 2 (pozice a orientace relativně k předdefinovanému nástroji tool0).

[\CalibFrame3]

Datový typ: pose

Číslo alternativního kalibračního rámce LATR 3 (pozice a orientace relativně k předdefinovanému nástroji tool0).

Syntaxe

```
CapLATrSetup
  [device ':='] < expression (IN) of string> ','
  [CalibFrame ':='] < persistent (PERS) of pose > ','
  [CalibPos ':='] < persistent (PERS) of pos >
  [\WarnMaxCorr]
  [\LogFile ':=' < expression (IN) of string >]
  [\LogSize ':=' < expression (IN) of num >]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
[\SensorFreq ':=' < expression (IN) of num >]
[\IpolServoDelay ':=' < expression (IN) of num >]
[\IpolCorrGain ':=' < expression (IN) of num >]
[\ServoSensFactor ':=' < expression (IN) of num >]
[\CorrFilter ':=' < expression (IN) of num >]
[\IpolCorrFilter ':=' < expression (IN) of num >]
[\ServoCorrFilter ':=' < expression (IN) of num >]
[\ErrRampIn ':=' < expression (IN) of num >]
[\ErrRampOut ':=' < expression (IN) of num >]
[\CBAngle ':=' < expression (IN) of num >]
[\MaxBlind ':=' < expression (IN) of num >]
[\MaxIncCorr ':=' < expression (IN) of num >]
[\CalibFrame2 ':=' < persistent (PERS) of pose >]
[\CalibFrame3 ':=' < persistent (PERS) of pose >] ';'
```

Související informace

Pro informace o	Viz
Sensor Interface	*Application manual - Controller software IRC5*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.31 CapNoProcess - Nechat běžet CAP bez procesu

Použití

`CapNoProcess` se používá k běhu CAP na určitou vzdálenost bez procesu.

S `CapNoProcess`, je možné přikázat CAP provést určitou vzdálenost (v mm) bez procesu. To je výhodné, jestliže se jednalo o odstranitelnou procesní chybu, která někdy neumožňuje restartovat proces v místě chyby.

Na začátku a na konci skokové vzdálenosti je podpora na dráze (`restart_dist` komponent v `capdata`) potlačena.

Na konci skokové vzdálenosti byla generována chyba s errno CAP_NOPROC_END.

Základní příklad

```
VAR num skip_dist := 0.0;
VAR bool cap_skip := FALSE;

PROC main()
  ......
  skip_dist := 25.0;
  CapL p_fig3_l_1, v500, cd, wsd, cwd, fine, tWeldGun;
  ......
  skip_dist := 15.0;
  CapL p_fig3_l_3, v500, cd, wsd, cwd, fine, tWeldGun;
  ......

  ERROR
    StorePath;
    TEST ERRNO
    CASE CAP_NOPROC_END:
      IF cap_skip THEN
        ! This is the end of the skip distance
        cap_skip := FALSE;
      ENDIF
    CASE CAP_MAIN_ERR:
      IF skip_dist > 0.0 THEN
        ! This is the start of the skip distance
        CapNoProcess skip_dist;
        cap_skip := TRUE;
      ENDIF
    DEFAULT:
    ENDTEST
    RestoPath;
    StartMoveRetry;
ENDPROC
ENDMODULE
```

V tomto příkladu je odstranitelná chyba CAP_MAIN_ERR následována 25 mm dlouhým pohybem (při 10 mm/s) bez procesu pro první instrukci `CapL` a při 15

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

mm/s pro druhý. Na konci této vzdálenosti je generován CAP_NOPROC_END a proces je restartován.

Argumenty

```
CapNoProcess skip_distance
```

skip_distance

Vzdálenost v mm

Datový typ: `num`

`CapNoProcess` má proměnnou `num` jako vstupní parametr, který definuje skokovou vzdálenost v mm.

Omezení

Rychlost TCP během přeskočení je předdefinována s 10 mm/s. Nejkratší vzdálenost přeskočení je předdefinována s 10 mm.

V synchro systémech MultiMove je nejkratší vzdálenost všech přeskokových vzdáleností definovaná pro roboty s odlišným sychro procesem ta aktuální.

Jestliže je vzdálenost přeskoku delší než vzdálenost od aktuální pozice TCP na konec aktuální sekvence instrukcí CAP, nestane se nic zvláštního: prováděníRAPID pokračuje jako obvykle, bez zastavení robotu.

Syntaxe

```
CapNoProcess
  [skip_dist ':='] < variable (IN) of num >';'
```

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instrukce `InitSuperv`	*InitSuperv - Resetovat veškerý dohled pro CAP na str 271*
Instrukce `SetupSuperv`	*SetupSuperv - Nastavit podmínky pro dohled signálu v CAP na str 639*
Instrukce `RemoveSuperv`	*RemoveSuperv - Odstranit podmínku pro jeden signál na str 538*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.32 CapRefresh - Obnovit CAP data

Použití

`CapRefresh` se používá k přikázání CAP procesu obnovit jeho procesní data. Může např. být použit pro ladění procesních parametrů CAP během vykonávání programu.

Základní příklad

```
PROC PulseSpeed()
  ! Setup a 1 Hz timer interrupt
  CONNECT intno1 WITH TuneTrp;
  ITimer 1, intno1;
  CapL p1, v100, cdata, wstartdata, wdata, fine, gun1;
  IDelete intno1;
ENDPROC

TRAP TuneTrp
  ! Modify the main speed component of active cdata
  IF HighValueFlag = TRUE THEN
    cdata.speed_data.start := 10;
    HighValueFlag := FALSE;
  ELSE
    cdata.speed_data.start := 15;
    HighValueFlag := TRUE;
  ENDIF
  ! Order the process control to refresh process parameters
  CapRefresh;
ENDTRAP
```

Na tomto příkladu bude rychlost přepínána mezi 10 a 15 mm/s při frekvenci 1 Hz.

Argumenty

```
CapRefresh [\MainSpeed] [MainWeave] [\StartWeave] [\RestartDist]
```

Bez volitelného argumentu jsou CPA data `capdata`, `capweavedata`, `weavestartdata`, `captrackdata`, a `movestarttimer` - pokud jsou přítomna - znovu načtena od PERSISTENT RAPID proměnné stanovené v aktuálně aktivní instrukci CAP.

[\MainSpeed]

Datový typ: `switch`

Jestliže je tento přepínač přítomný, CAP znovu načte komponent `capdata.speed_data.main` aktuálně aktivní CAP instrukce.

[\MainWeave]

Datový typ: `switch`

Jestliže je tento přepínač přítomný, CAP znovu načte komponenty `capweavedata.width`, `capweavedata. length`, `capweavedata.bias`, a `capweavedata.active` aktuálně aktivní CAP instrukce.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

[\StartWeave]

Datový typ: `bool`

Jestliže je tento přepínač přítomný, CAP použije jeho hodnoty místo `weavestartdata.active` aktuálně aktivní CAP instrukce. Data aktuálně aktivní CAP instrukce zůstanou nedotčena.

[\RestartDist]

Datový typ: `num`

Jestliže je tento přepínač přítomný, CAP použije jeho hodnoty místo `weavestartdata.active` aktuálně aktivní CAP instrukce. Data aktuálně aktivní CAP instrukce zůstanou nedotčena.

Syntaxe

```
CapRefresh
  ['\' MainSpeed]
  ['\' MainWeave]
  ['\' Startweave ':=' < expression (IN) of bool >]
  ['\' RestartDist ':=' < expression (IN) of num >] ';'
```

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.33 CapWeaveSync - signály nastavení a úrovní pro weave synchronizaci

Použití

`CapWeaveSync` se používá k nastavení signálů weaving synchronizace bez senzorů. I/O signály musí být definovány v EIO.cfg.

Základní příklad

RAPID program:

```
PROC main()
  ...
  CapWeaveSync \DoLeft:=do_sync_left \LevelLeft:=80
       \DoRight:=do_sync_right \LevelRight:=80;
  ...
ENDPROC
```

Na tomto příkladu jsou signály `do_sync_left and do_sync_right` nastaveny s weaving úrovní 80 %.

Instrukce `CapWeaveSync` by měla být vykonána pouze jednou, například ze zakládací přihrádky.

Argumenty

```
CapWeaveSync [\Reset] [\DoLeft] [\LevelLeft] [\DoRight]
     [\LevelRight]
```

[\Reset]

Datový typ: `switch`

Vyčistit synchronizační data weave.

[\DoLeft]

Datový typ: `signaldo`

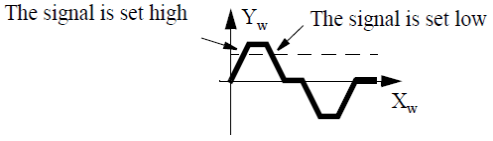
Digitální výstupní signál pro weave synchronizaci na levém weave cyklu.

[\LevelLeft]

Datový typ: `num`

Koordinační pozice na levé straně weaving obrazce. Určená hodnota je procentní částí šířky vlevo od weaving středu. Když je weaving prováděn za tímto bodem, digitální výstupní signál se automaticky nastaví vysoko (za předpokladu, že signál je definován).

Tento typ koordinace se může používat pro sledování spoje pomocí Through-the-Arc Tracker.



xx1200000176

[\LevelLeft]

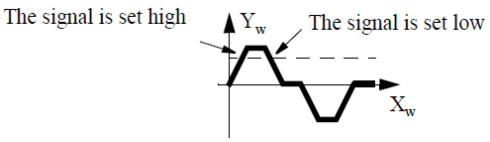
Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Koordinační pozice na levé straně weaving obrazce. Určená hodnota je procentní částí šířky vlevo od weaving středu. Když je weaving prováděn za tímto bodem, digitální výstupní signál se automaticky nastaví vysoko (za předpokladu, že signál je definován).

Tento typ koordinace se může používat pro sledování spoje pomocí Through-the-Arc Tracker.



xx1200000176

[\DoRight]

Datový typ: `signaldo`

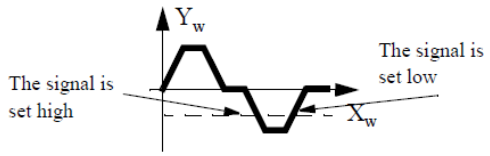
Digitální výstupní signál pro weave synchronizaci na pravém weave cyklu.

[\LevelRight]

Datový typ: `num`

Koordinační pozice na pravé straně weaving obrazce. Určená hodnota je procentní částí šířky vpravo od weaving středu. Když je weaving prováděn za tímto bodem, digitální výstupní signál se automaticky nastaví vysoko (za předpokladu, že signál je definován).

Tento typ koordinace se může používat pro sledování spoje pomocí Through-the-Arc Tracker.



xx1200000177

Vykonávání programu

Definované signály jsou kontrolovány a nastaveny při běhu bez senzoru.

Omezení

Signály musí být definovány v EIO.cfg.

Není možné používat pouze buď úroveň nebo odpovídající signál. Výsledkem nebudou chyby při načítání souboru RAPID , ale povede to chybám při běhu RAPID pro instrukci `CapWeaveSynch`.

Syntaxe

```
CapWeaveSync
  ['\' Reset]
  [DoLeft ':=' < expression (IN) of signaldo >]
  [LevelLeft ':=' < expression (IN) of num >]
  [DoRight ':=' < expression (IN) of signaldo >]
  [LevelRight ':=' < expression (IN) of num >] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.33 CapWeaveSync - signály nastavení a úrovní pro weave synchronizaci

Continuous Application Platform (CAP)

Pokračování

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Datový typ `capweavedata`	*capweavedata - Weavedata pro CAP na str 1463*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.34 CheckProgRef - Zkontrolovat reference programu

Použití

`CheckProgRef` se používá pro kontrolu nevyřešených referencí kdykoliv během vykonávání.

Základní příklady

Následující příklad názorně ukazuje instrukci `CheckProgRef`:

Příklad 1

```
Load \Dynamic, diskhome \File:="PART_B.MOD" \CheckRef;
Unload "PART_A.MOD";
CheckProgRef;
```

V tomto případě program obsahuje modul nazvaný `PART_A.MOD`. Nový modul `PART_B.MOD` je načten, což je kontrola, jestli všechny reference jsou OK. Potom je vymazán `PART_A.MOD`. Kvůli kontrole nevyřešených referencí po vymazání je provedeno volání k `CheckProgRef`.

Vykonávání programu

Vykonávání programu si vynucuje novou řádku programové úlohy a kontroluje nevyřešené reference.

Jestliže se objeví chyba během `CheckProgRef`, program není ovlivněn, pouze vám oznámí, že v programové úloze existuje nevyřešená reference. Proto použijte `CheckProgRef` okamžitě po změně čísla modulů v programové úloze (načítání a stažení), aby bylo možné zjistit, který modul působí chybu řádky.

Tuto instrukci je také možné používat jako náhradu za používání volitelného argumentu `\CheckRef` v instrukci `Load` nebo `WaitLoad`.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_LINKREF`	Programová úloha obsahuje nevyřešené reference.

Syntaxe

```
CheckProgRef';'
```

Související informace

Pro informace o	Viz
Načíst programový modul	*Load - Načíst programový modul během provádění na str 328*
Vymazat data z programového modulu	*UnLoad - Stáhnout programový modul během provádění na str 905*
Začít načítat programový modul	*StartLoad - Načíst programový modul během vykonávání na str 703*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.34 CheckProgRef - Zkontrolovat reference programu

RobotWare - OS

Pokračování

Pro informace o	Viz
Ukončit načítání programového modulu	*WaitLoad - Připojit načtený modul k úloze na str 947*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.35 CirPathMode - Reorientace nástroje během kruhové dráhy

Použití

`CirPathMode` (*Circle Path Mode*) umožňuje zvolit různé režimy k reorientaci nástroje během kruhových pohybů.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `CirPathMode`:

Příklad 1

```
CirPathMode \PathFrame;
```

Standardním režimem pro reorientaci nástroje v rámci skutečné dráhy od bodu startu k `ToPoint` během všech následných kruhových pohybů. To je výchozí stav v systému.

Příklad 2

```
CirPathMode \ObjectFrame;
```

Upravený režim pro reorientaci nástroje v rámci skutečného objektu od bodu startu k `ToPoint` během všech následných kruhových pohybů.

Příklad 3

```
CirPathMode \CirPointOri;
```

Upravený režim pro reorientaci nástroje od bodu startu přes naprogramovanou orientaci `CirPoint` k `ToPoint` během všech následných kruhových pohybů.

Příklad 4

```
CirPathMode \Wrist45;
```

Upravený režim, jako je projekce Z osy nástroje na rovinu řezu, bude následovat pořadí naprogramovaného kruhového pohybu. Používají se pouze osy zápěstí 4 a 5. Tento režim by se měl používat pouze pro tenké objekty.

Příklad 5

```
CirPathMode \Wrist46;
```

Upravený režim, jako je projekce Z osy nástroje na rovinu řezu, bude následovat pořadí naprogramovaného kruhového pohybu. Používají se pouze osy zápěstí 4 a 6. Tento režim by se měl používat pouze pro tenké objekty.

Příklad 6

```
CirPathMode \Wrist56;
```

Upravený režim, jako je projekce Z osy nástroje na rovinu řezu, bude následovat pořadí naprogramovaného kruhového pohybu. Používají se pouze osy zápěstí 5 a 6. Tento režim by se měl používat pouze pro tenké objekty.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.35 CirPathMode - Reorientace nástroje během kruhové dráhy

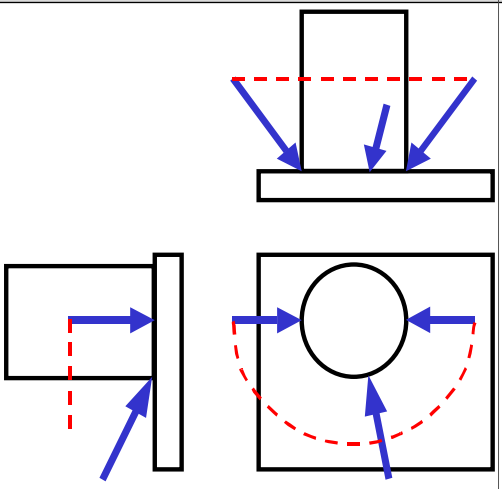
RobotWare - OS

Pokračování

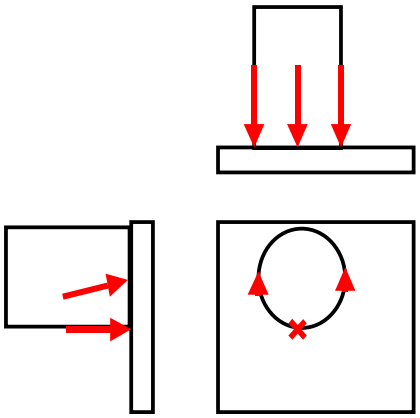
Popis

PathFrame

Číslo (obrázek) v tabulce ukazuje reorientaci nástroje pro standardní režim `\PathFrame`.

Obrázek	Popis
	Šipky ukazují nástroj od středního bodu zápěstí ke střednímu bodu nástroje pro naprogramované body. Dráha pro středový bod zápěstí je na obrázku tečkovaná. Režim `\PathFrame` usnadňuje získat stejný úhel nástroje kolem válce. Zápěstí robotu nebude procházet naprogramovanou orientací v `CirPoint`

Obrázek v tabulce ukazuje použití standardního režimu `\PathFrame` s orientací pevného nástroje.

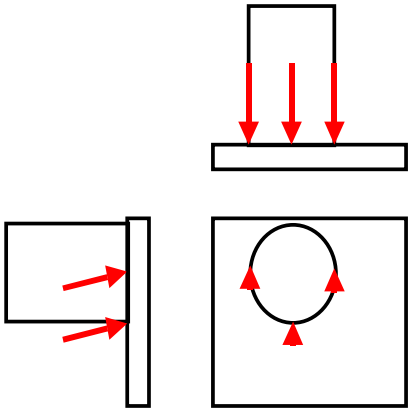
Obrázek	Popis
	Tento obrázek ukazuje získanou orientaci nástroje uprostřed kruhu pomocí nakloněného nástroje a režimu `\PathFrame`. Srovnejte s obrázkem dole, když je použit režim `\ObjectFrame`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

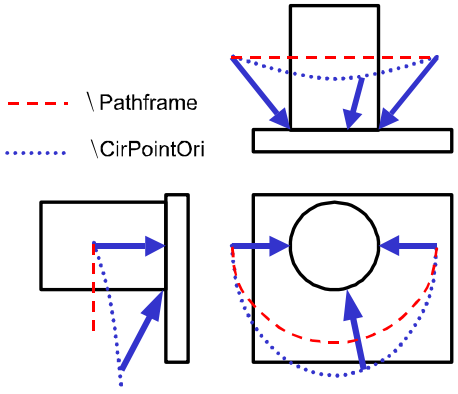
ObjectFrame

Obrázek v tabulce ukazuje použití modifikovaného režimu `\ObjectFrame` s orientací pevného nástroje.

Obrázek	Popis
 xx0500002151	Tento obrázek ukazuje získanou orientaci nástroje uprostřed kruhu pomocí nakloněného nástroje a režimu `\ObjectFrame`. Tento režim zajistí lineární reorientaci nástroje stejně jako `MoveL`. Zápěstí robotu nebude procházet naprogramovanou orientací v `CirPoint`. Srovnejte s předchozím obrázkem, když je použit režim `\PathFrame`.

CirPointOri

Číslo (obrázek) v tabulce ukazuje různou reorientaci nástroje mezi standardním režimem `\PathFrame` a modifikovaným režimem `\CirPointOri`.

Obrázek	Popis
 xx0500002150	Šipky ukazují nástroj od středního bodu zápěstí ke střednímu bodu nástroje pro naprogramované body. Různé dráhy pro středový bod zápěstí jsou na obrázku čárkované. Režim `\CirPointOri` zajistí průchod zápěstí robotu naprogramovanou orientací v `CirPoint`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

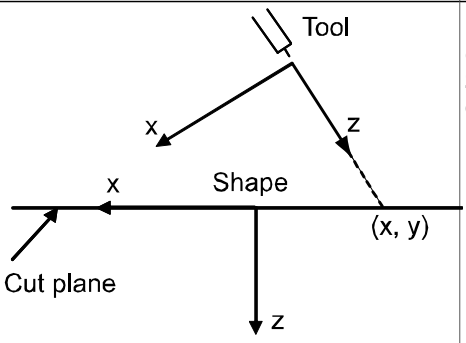
1.35 CirPathMode - Reorientace nástroje během kruhové dráhy

RobotWare - OS

Pokračování

Wrist45 / Wrist46 / Wrist56

Obrázek v tabulce ukazuje zahrnuté rámce při řezání tvaru pomocí osy 4 a 5. .

Obrázek	Popis
	Předpokládá se, že řezný paprsek je slícován s osou Z nástroje. Souřadnicový rámec roviny řezu je definován startovní pozicí robotu při provádění instrukce MoveC.

Argumenty

```
CirPathMode [\PathFrame] | [\ObjectFrame] | [\CirPointOri] |
     [\Wrist45] | [\Wrist46] | [\Wrist56]
```

`[ \PathFrame ]`

Datový typ: `switch`

Během kruhového pohybu je reorientace nástroje prováděna nepřetržitě od orientace bodu startu k orientaci `ToPoint` ve skutečném rámci dráhy. To je standardní režim v systému.

`[ \ObjectFrame ]`

Datový typ: `switch`

Během kruhového pohybu je reorientace nástroje prováděna nepřetržitě od orientace bodu startu k orientaci `ToPoint` ve skutečném rámci dráhy.

`[ \CirPointOri ]`

Datový typ: `switch`

Během kruhového pohybu je reorientace nástroje prováděna nepřetržitě od orientace bodu startu k naprogramované orientaci `CirPoint` a dále k orientaci `ToPoint`.

`[ \Wrist45 ]`

Datový typ: `switch`

Robot posune osy 4 a 5 tak, že projekce osy Z nástroje na rovinu řezu bude sledovat naprogramovaný sled kruhových pohybů. Tento režim by se měl používat pouze pro slabé objekty, protože jsou použity pouze dvě osy zápěstí, což nám dává zvýšenou přesnost, ale také méně kontroly.

POZNÁMKA

Tento přepínač vyžaduje doplněk Wrist Move.

`[ \Wrist46 ]`

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Robot posune osy 4 a 6 tak, že projekce osy Z nástroje na rovinu řezu bude sledovat naprogramovaný sled kruhových pohybů. Tento režim by se měl používat pouze pro slabé objekty, protože jsou použity pouze dvě osy zápěstí, což nám dává zvýšenou přesnost, ale také méně kontroly.

POZNÁMKA

Tento přepínač vyžaduje doplněk Wrist Move.

`[ \Wrist56 ]`

Datový typ: `switch`

Robot posune osy 5 a 6 tak, že projekce osy Z nástroje na rovinu řezu bude sledovat naprogramovaný sled kruhových pohybů. Tento režim by se měl používat pouze pro slabé objekty, protože jsou použity pouze dvě osy zápěstí, což nám dává zvýšenou přesnost, ale také méně kontroly.

Jestliže používáte `CirPathMode` bez přepínače, potom je výsledek stejný jako `CirPointMode \PathFrame`

POZNÁMKA

Tento přepínač vyžaduje doplněk Wrist Move.

Vykonávání programu

Stanovený reorientační režim kruhového nástroje se vztahuje na další vykonané kruhové pohyby robotu jakéhokoliv typu (`MoveC, SearchC, TriggC, MoveCDO, MoveCSync, ArcC, PaintC` ...) a je platný až do provedení nové instrukce `CirPathMode` (nebo překonané `CirPathReori`).

Standardní kruhový režim reorientace (`CirPathMode \PathFrame`) se nastavuje automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.35 CirPathMode - Reorientace nástroje během kruhové dráhy

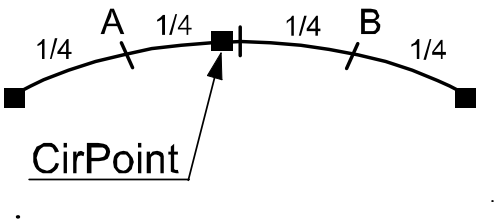
RobotWare - OS

Pokračování

Omezení

Instrukce ovlivňuje pouze kruhové pohyby.

Při použití režimu `\CirPointOri` musí být `CirPoint` mezi body **A** a **B** podle obrázku dole, aby bylo možné udělat kruhový pohyb **skrz** naprogramovanou orientaci v `CirPoint`.



xx0500002149

Režim `\Wrist45`, `\Wrist46`, a `\Wrist56` by se měl používat pouze pro řezání tenkých objektů, protože schopnost kontroly úhlu nástroje je ztracena, když se používají pouze dvě osy zápěstí. Koordinované pohyby nejsou možné, protože hlavní osa je zablokována.

Při práci v oblasti singularity zápěstí, když byla vykonána instrukce `SingArea\Wrist`, nemá instrukce `CirPathMode` žádný účinek, protože systém potom vybere jiný režim reorientace nástroje pro kruhové pohyby (interpolace spoje).

Tato instrukce nahrazuje starou instrukci `CirPathReori` (bude fungovat i v budoucnosti, ale už nebude zdokumentována).

Syntaxe

```
CirPathMode
  ['\'PathFrame]
  | ['\'ObjectFrame]
  | ['\'CirPointOri]
  | ['\'Wrist45]
  | ['\'Wrist46]
  | ['\'Wrist56] ';'
```

Související informace

Pro informace o	Viz
Interpolace	*Technická referenční příručka - Přehled RAPID*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*
Instrukce kruhového pohybu	*MoveC - Pohybuje robotem kruhově na str 358*
Pohyby zápěstí	*Application manual - Controller software IRC5*, sekce *Pohyb zápěstí*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.36 Vyčistit - Vyčistí hodnotu

Použití

`Clear` se používá pro vyčištění numerické proměnné nebo perzistentu, to znamená, nastaví je na 0.

Základní příklady

Následující příklady názorně ukazují instrukci `Clear`:

Příklad 1

```
Clear reg1;
```

`Reg1` je vyčištěn, tj. `reg1:=0`.

Příklad 2

```
CVAR dnum mydnum:=5;
Clear mydnum;
```

`mydnum` je vyčištěn, tj. `mydnum:=0`.

Argumenty

```
Clear Name | Dname
```

`Name`

Datový typ: `num`

Jméno proměnné nebo perzistentu, které budou vyčištěny.

`Dname`

Datový typ: `dnum`

Jméno proměnné nebo perzistentu, které budou vyčištěny.

Syntaxe

```
Clear
  [ Name ':=' ] < var or pers (INOUT) of num >
  | [ Dname ':=' ] < var or pers (INOUT) of dnum > ';'
```

Související informace

Pro informace o	Viz
Přírůstkování proměnné o 1	*Incr - Přírůstky po 1 na str 251*
Snižování proměnné o 1	*Decr - Snižování po 1 na str 152*
Přidání jakékoliv hodnoty k proměnné	*Add - Přidává numerickou hodnotu na str 26*
Změna dat pomocí libovolného	*":=" - Přiděluje hodnotu na str 33*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.37 ClearIOBuff - Vyčistit vstupní zásobník sériového kanálu

Použití

`ClearIOBuff` (*Clear I/O Buffer*) se používá k vyčištění vstupního zásobníku sériového kanálu. Všechny znaky vstupního sériového kanálu jsou ze zásobníku vyřazeny.

Základní příklady

Následující příklad názorně ukazuje instrukci `ClearIOBuff`:

Příklad 1

```
VAR iodev channel1;
...
Open "com1:", channel1 \Bin;
ClearIOBuff channel1;
WaitTime 0.1;
```

Vstupní zásobník pro sériový kanál odkazovaný od `channel1` je vyčištěn. Doba čekání zaručuje dostatek času pro dokončení vyčišťovací operace.

Argumenty

```
ClearIOBuff IODevice
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) sériového kanálu, jehož vstupní zásobník bude vyčištěn.

Vykonávání programu

Všechny znaky v zásobníku ze vstupního sériového kanálu jsou vyřazeny. Další instrukce pro načtení budou čekat na nový vstup z kanálu.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Omezení

Tuto instrukci je možné používat pouze u sériových kanálů. Nečekejte na potvrzení o dokončení operace. Umožněte dobu čekání 0.1 poté, co je instrukce doporučena, aby poskytla operaci dostatek času v každé aplikaci.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_FILEACC`	Instrukce se používá na souboru.

Syntaxe

```
ClearIOBuff
  [IODevice ':='] <variable (VAR) of iodev>';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Otevření sériového kanálu	*Technická referenční příručka - Přehled RAPID*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.38 ClearPath - Vyčistit aktuální dráhu

Použití

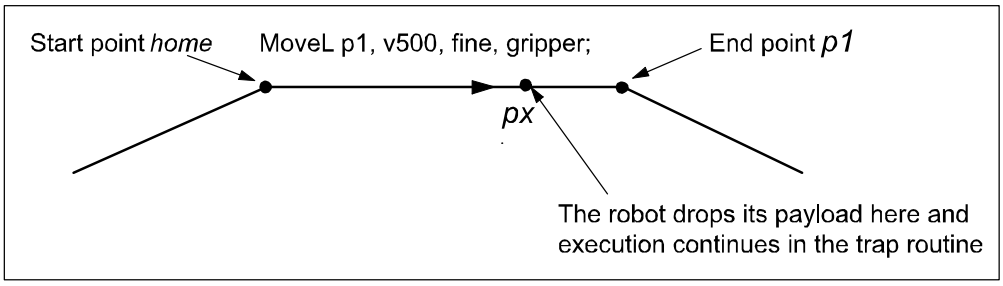
`ClearPath` (*Clear Path*) vyčistí celou dráhu pohybu na aktuální úrovni dráhy pohybu (základní úroveň nebo úroveň `StorePath`.

Dráhou pohybu se rozumí všechny příkazy k pohybu od všech pohybových instrukcí, které byly vykonány v RAPIDu, ale nebyly provedeny robotem v době vykonávání `ClearPath`.

Robot musí být v pozici stop bodu nebo musí být zastaven s `StopMove` předtím, než může být vykonána instrukce `ClearPath`.

Základní příklady

Následující příklad názorně ukazuje instrukci `ClearPath`:



xx0500002154

V následujícím programovému příkladu se robot pohybuje od pozice `home` k pozici `p1`. V bodě `px` bude signál `di1` indikovat, že užitečné zatížení bylo odhozeno. Vykonávání pokračuje v trap rutině `gohome`. Robot zastaví pohyb (začne brzdit) na `px`, dráha bude vyčištěna, robot se posune do pozice `home`. Chyba bude vyvolána k volající rutině `minicycle` a celý uživatelsky definovaný programový cyklus `proc1 ... proc2` bude vykonán od začátku ještě jednou.

Příklad 1

```
VAR intnum drop_payload;
VAR errnum ERR_DROP_LOAD := -1;

PROC minicycle()
  BookErrNo ERR_DROP_LOAD;
  proc1;
  ...
  ERROR (ERR_DROP_LOAD)
    ! Restart the interupted movement on motion base path level
    StartMove;
    RETRY;
ENDPROC

PROC proc1()
  ...
  proc2;
  ...
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
ENDPROC

PROC proc2()
  CONNECT drop_payload WITH gohome;
  ISignalDI \Single, di1, 1, drop_payload;
  MoveL p1, v500, fine, gripper;
  ...........
  IDelete drop_payload;
ENDPROC

TRAP gohome
  StopMove \Quick;
  ClearPath;
  IDelete drop_payload;
  StorePath;
  MoveL home, v500, fine, gripper;
  RestoPath;
  RAISE ERR_DROP_LOAD;
  ERROR
    RAISE;
ENDTRAP
```

Jestliže stejný program běží, ale bez`StopMove` **a** `ClearPath` **v trap rutině** `gohome`**, robot bude pokračovat na pozici** `p1` **předtím, než se vrátí zpět na pozici** `home`**.**

Omezení

Příklady omezení instrukce `ClearPath` **jsou názorně uvedeny dole.**

Příklad 1 - Omezení

```
VAR intnum int_move_stop;
...
PROC test_move_stop()
  CONNECT int_move_stop WITH trap_move_stop;
  ISignalDI di1, 1, int_move_stop;
  MoveJ p10, v200, z20, gripper;
  MoveL p20, v200, z20, gripper;
ENDPROC

TRAP trap_move_stop
  StopMove;
  ClearPath;
  StorePath;
  MoveJ p10, v200, z20, gripper;
  RestoPath;
  StartMove;
ENDTRAP
```

Toto je příklad omezení `ClearPath`**. Během pohybu robotu k** `p10` **a** `p20` **je probíhající pohyb zastaven a dráha pohybu je vyčištěna, ale není provedena žádná činnost k ukončení aktivní instrukce** `MoveJ p10` **nebo** `MoveL p20` **v** `PROC test_move_stop`**. Takže probíhající pohyb bude přerušen a robot přejde k** `p10` **v**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.38 ClearPath - Vyčistit aktuální dráhu
Robot Ware - OS
Pokračování

TRAP trap_move_stop, ale žádný další pohyb k p10 nebo p20 v PROC test_move_stop nebude proveden. Vykonávání programu bude zastaveno.

Tento problém je možné vyřešit buď obnovou po chybě s dlouhým skokem, jak je popsáno v příkladu 2 dole, nebo s asynchronně vyvolanou chybou s instrukcí ProcerrRecovery.

Příklad 2 - Bez omezení

```
VAR intnum int_move_stop;
VAR errnum err_move_stop := -1;
...
PROC test_move_stop()
  BookErrNo err_move_stop;
  CONNECT int_move_stop WITH trap_move_stop;
  ISignalDI di1, 1, int_move_stop;
  MoveJ p10, v200, z20, gripper;
  MoveL p20, v200, z20, gripper;
  ERROR (err_move_stop)
    StopMove;
    ClearPath;
    StorePath;
    MoveJ p10, v200, z20, gripper;
    RestoPath;
    ! Restart the interupted movement on motion base path level
    StartMove;
    RETRY;
ENDPROC

TRAP trap_move_stop
  RAISE err_move_stop;
  ERROR
    RAISE;
ENDTRAP
```

Toto je příklad, jak používat obnovu po chybě s dlouhým skokem společně s ClearPath bez jakéhokoliv omezení. Během pohybu robotu k p10 a p20 je probíhající pohyb zastaven. Dráha pohybu je vyčištěna, ale protože se jedná o obnovu po chybě přes hranice úrovně vykonávání, je provedeno přerušení aktivní instrukce MoveJ p10 nebo MoveL p20. Takže probíhající pohyb bude přerušen a robot přejde k p10 v ERROR handler a ještě jednou vykoná přerušenou instrukci MoveJ p10 nebo MoveL p20 v PROC test_move_stop.

Syntaxe

```
ClearPath ';'
```

Související informace

Pro informace o	Viz
Zastavit pohyby robotu	*StopMove - Zastavuje pohyby robotu na str 736*
Obnovení po chybě	*Technická referenční příručka - Přehled RAPID* *Technická referenční příručka - RAPID kernel*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Asynchronně vyvolaná chyba	*ProcerrRecovery - Generovat a zotavit z chyby procesního pohybu na str 485*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.39 ClearRawBytes - Vyčistit obsahy rawbyte dat

Použití

`ClearRawBytes` se používá k nastavení všech obsahů `rawbytes` proměnné na 0.

Základní příklady

Následující příklad názorně ukazuje instrukci `ClearRawBytes`:

Příklad 1

```
VAR rawbytes raw_data;
VAR num integer := 8
VAR num float := 13.4;

PackRawBytes integer, raw_data, 1 \IntX := DINT;
PackRawBytes float, raw_data, (RawBytesLen(raw_data)+1) \Float4;

ClearRawBytes raw_data \FromIndex := 5;
```

V prvních 4 bytech je umístěna hodnota `integer` (od indexu 1) a v dalších 4 bytech od indexu 5 hodnota `float`.

Poslední instrukce v příkladu vyčistí obsah `raw_data`, začne na indexu 5, to znamená, že `float` bude vyčištěn, ale `integer` bude zachován v `raw_data`.

Aktuální délka platných bytů v `raw_data` je nastavena na 4.

Argumenty

```
ClearRawBytes RawData [ \FromIndex ]
```

`RawData`

Datový typ: `rawbytes`

`RawData` je datový kontejner, který bude vyčištěn.

`[ \FromIndex ]`

Datový typ: `num`

`\FromIndex` určuje, kde začít s čištěním obsahu `RawData`. Všechno bude vyčištěno až do konce.

Jestliže není stanoveno `\FromIndex`, veškerá data od indexu 1 budou vyčištěna.

Vykonávání programu

Data od indexu 1 (standard) nebo od `\FromIndex` v určené proměnné jsou resetována na 0.

Aktuální délka platných bytů v určené proměnné je nastavena na 0 (standard) nebo na (`FromIndex` - 1), jestliže je naprogramován `\FromIndex`.

Syntaxe

```
ClearRawBytes
  [RawData ':=' ] < variable (VAR) of rawbytes>
  ['\'FromIndex ':=' <expression (IN) of num>]';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Data`rawbytes`	*rawbytes - Data raw na str 1559*
Získat délku dat `rawbytes`	*RawBytesLen - Získat délku dat rawbytes na str 1278*
Kopírovat obsah dat `rawbytes`	*CopyRawBytes - Kopírovat obsah dat rawbytes na str 137*
Zabalit hlavičku DeviceNet do dat `rawbytes`	*PackDNHeader - Zabalit hlavičku DeviceNet do dat rawbytes na str 446*
Zabalit data do dat `rawbytes`	*PackRawBytes - Zabalit data do dat rawbytes na str 449*
Zapsat data `rawbytes`	*WriteRawBytes - Zapsat data rawbytes na str 994*
Načíst data `rawbytes`	*ReadRawBytes - Přečíst data rawbytes na str 530*
Rozbalit data z dat `rawbytes`	*UnpackRawBytes - Rozbalit data z rawbyte dat na str 908*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.40 ClkReset - Resetuje hodiny používané pro časování

Použití

`ClkReset` se používá k resetu hodin, které fungují jako stopky pro časování.

Tuto instrukci je možné používat před použitím hodin, aby bylo jisté, že jsou nastaveny na 0.

Základní příklady

Následující příklad názorně ukazuje instrukci `ClkReset`:

Příklad 1

```
ClkReset clock1;
```

Hodiny `clock1` jsou resetovány.

Argumenty

```
ClkReset Clock
```

`Clock`

Datový typ: `clock`

Jméno hodin, které budou resetovány.

Vykonávání programu

Při restartu hodin je hodnota nastavena na 0.

Jestliže hodiny běží, budou zastaveny a potom resetovány.

Syntaxe

```
ClkReset
  [ Clock ':=' ] < variable (VAR) of clock > ';'
```

Související informace

Pro informace o	Viz
Další instrukce k hodinám	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.41 ClkReset - Spustí hodiny používané pro časování

Použití

`ClkStart` se používá ke spuštění hodin, které fungují jako stopky používané pro časování.

Základní příklady

Následující příklad názorně ukazuje instrukci `ClkStart`:

Příklad 1

```
ClkStart clock1;
```

Hodiny `clock1` jsou spuštěny.

Argumenty

```
ClkStart Clock
```

`Clock`

Datový typ: `clock`

Jméno hodin, které budou spuštěny.

Vykonávání programu

Když jsou hodiny spuštěny, poběží a budou počítat sekundy, dokud nebudou zastaveny.

Hodiny dále běží a když se spustí program, zastaví se. Nicméně, událost, kterou jste měli v úmyslu stopnout, už nemusí být nadále platná. Například, jestliže program měřil čekací čas pro vstup, vstup mohl být přijat, zatímco program byl zastaven. V tomto případě nebude program schopen „vidět“ událost, která vznikla, zatímco program byl zastaven.

Hodiny stále běží, když je robot vypnut, dokud záloha baterií udržuje program, který obsahuje proměnnou hodin.

Jestliže hodiny běží, je možné je sledovat, zastavit nebo resetovat.

Další příklady

Více příkladů instrukce `ClkStart` je názorně uvedeno dole.

Příklad 1

```
VAR clock clock2;
VAR num time;

ClkReset clock2;
ClkStart clock2;
WaitUntil di1 = 1;
ClkStop clock2;
time:=ClkRead(clock2);
```

Čekací čas, až `di1` bude 1, je měřen.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.41 ClkReset - Spustí hodiny používané pro časování

RobotWare - OS

Pokračování

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_OVERFLOW.	Hodiny běží 4 294 967 sekund (49 dní 17 hodin 2 minuty 47 sekund), potom dojde k přeplnění.

Syntaxe

```
ClkStart
  [ Clock ':=' ] < variable (VAR) of clock >';'
```

Související informace

Pro informace o	Viz
Další instrukce k hodinám	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.42 ClkReset - Zastaví hodiny používané pro časování

Použití

`ClkStop` se používá k zastavení hodin, které fungují jako stopky používané pro časování.

Základní příklady

Následující příklad názorně ukazuje instrukci `ClkStop`:

```
ClkStop clock1;
```

Hodiny `clock1` byly zastaveny.

Argumenty

```
ClkStop Clock
```

`Clock`

Datový typ: `clock`

Jméno hodin, které budou zastaveny.

Vykonávání programu

Když jsou hodiny zastaveny, zastaví se běh.

Jestliže hodiny jsou zastaveny, je možné je sledovat, znovu spustit nebo resetovat.

Řešení chyb

Jestliže hodiny běží 4 294 967 sekund (49 dní 17 hodin 2 minuty 47 sekund), dojde k jejich přeplnění a systémová proměnná `ERRNO` je nastavena na `ERR_OVERFLOW`.

Chyba může být zpracována v handleru chyb.

Syntaxe

```
ClkStop
[ Clock ':=' ] < variable (VAR) of clock >';'
```

Související informace

Pro informace o	Viz
Další instrukce k hodinám	*Technická referenční příručka - Přehled RAPID*
Další příklady	*ClkReset - Spustí hodiny používané pro časování na str 121*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.43 Zavřít - Zavírá soubor nebo sériový kanál

Použití

`Close` se používá pro zavření souboru nebo sériového kanálu.

Základní příklady

Následující příklad názorně ukazuje instrukci `Close`:

Příklad 1

```
Close channel2;
```

Sériový kanál odkazovaný od `channel2` je uzavřen.

Argumenty

```
Close IODevice
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) souboru nebo sériového kanálu, který bude zavřen.

Vykonávání programu

Stanovený soubor nebo sériový kanál je zavřen a musí být znovu otevřen před načítáním nebo zápisem. Jestliže je už zavřen, instrukce je ignorována.

Syntaxe

```
Close
  [IODevice ':='] <variable (VAR) of iodev>';'
```

Související informace

Pro informace o	Viz
Otevření souboru nebo sériového kanálu	*Technická referenční příručka - Přehled RAPID*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.44 CloseDir - Zavřít adresář

Použití

`CloseDir` se používá k zavření adresáře v rovnováze s `OpenDir`.

Základní příklady

Následující příklad názorně ukazuje instrukci `CloseDir`:

Příklad 1

```
PROC lsdir(string dirname)
  VAR dir directory;
  VAR string filename;
  OpenDir directory, dirname;
  WHILE ReadDir(directory, filename) DO
    TPWrite filename;
  ENDWHILE
CloseDir directory;
ENDPROC
```

Tento příklad vytiskne jména všech souborů nebo podadresářů pod určeným adresářem.

Argumenty

```
CloseDir Dev
```

`Dev`

Datový typ: `dir`

Proměnná s referencí k adresáři dodaná s instrukcí `OpenDir`.

Syntaxe

```
CloseDir
  [ Dev ':=' ] < variable (VAR) of dir>';'
```

Související informace

Pro informace o	Viz
Adresář	*dir - Struktura adresáře souborů na str 1484*
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Otevřít adresář	*OpenDir - Otevřít adresář na str 444*
Načíst adresář	*ReadDir - Přečíst další vstupní data v adresáři na str 1283*
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.45 Comment - Komentář

Použití

`Comment` se používá pouze pro zjednodušení srozumitelnosti programu. Nemá žádný vliv na vykonávání programu.

Základní příklady

Následující příklad názorně ukazuje instrukci `Comment`:

Příklad 1

```
! Goto the position above pallet
MoveL p100, v500, z20, tool1;
```

Komentář se vkládá do programu, aby byl srozumitelnější.

Argumenty

```
! Comment
```

`Comment`

Textový řetězec

Jakýkoliv text.

Vykonávání programu

Když provedete tuto instrukci, nic se nestane.

Syntaxe

```
'!' {<character>} <newline>
```

Související informace

Pro informace o	Viz
Znaky povolené v komentáři	*Technická referenční příručka - Přehled RAPID*
Komentáře v deklaracích dat a rutin	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.46 Compact IF - Jestliže je splněna podmínka, potom... (jedna instrukce)

Použití

`Compact IF` se používá, když jednoduchá instrukce má být provedena pouze v případě, že je splněna daná podmínka.

Jestliže mají být vykonány odlišné instrukce v závislosti na tom, jestli je určená podmínka splněna nebo nikoliv, použije se instrukce `IF`.

Základní příklady

Následující příklady názorně ukazují instrukci `CompactIF`:

Příklad 1

```
IF reg1 > 5 GOTO next;
```

Jestliže `reg1` je větší než `5`, vykonávání programu pokračuje na návěstí `next`.

Příklad 2

```
IF counter > 10 Set do1;
```

Signál `do1` se nastavuje, když `counter > 10`.

Argumenty

```
IF Condition ...
```

Condition

Datový typ: `bool`

Podmínka, která musí být splněna, aby instrukce mohla být vykonána.

Syntaxe

```
IF <conditional expression> ( <instruction> | <SMT>) ';'
```

Související informace

Pro informace o	Viz
Podmínky (logické výrazy	*Technická referenční příručka - Přehled RAPID*
IF s několika instrukcemi	*IF - Jestliže je splněna podmínka, potom ...; jinak ... na str 249*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.47 ConfJ - Kontroluje konfiguraci během pohybu spoje

Použití

`ConfJ` (*Configuration Joint*) se používá k určení, jestli konfigurace robotu bude kontrolována během pohybu spoje nebo nikoliv. Jestliže není kontrolována, robot může někdy použít odlišnou konfiguraci, než která byla naprogramována.

S `ConfJ \Off` nemůže robot přepnout konfiguraci hlavní osy - bude hledat řešení se stejnou konfigurací hlavní osy jako aktuální, ale posune se k nejbližší konfiguraci zápěstí pro osy 4 a 6.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `ConfJ`:

Příklad 1

```
ConfJ \Off;
MoveJ *, v1000, fine, tool1;
```

Robot se posune k naprogramované pozici a orientaci. Jestliže je možné této pozice dosáhnout několika různými způsoby, je zvolena nejbližší možná pozice.

Příklad 2

```
ConfJ \On;
MoveJ *, v1000, fine, tool1;
```

Robot se posune na naprogramovanou pozici, orientaci a konfiguraci osy.

Argumenty

```
ConfJ [\On] | [\Off]
```

`[ \On ]`

Datový typ: `switch`

Robot se posune na naprogramovanou pozici s konfiguračními parametry totožnými nebo blízkými daným konfiguračním parametrům v `confdata`.

Jestliže je aktivní nahrazení programu nebo korekce dráhy, zvyšuje se riziko velkých pohybů, jelikož naprogramovaná konfigurační data jsou založena na původní pozici.

Robot IRB5400 se posune na naprogramovanou konfiguraci osy nebo na osu blízkou naprogramované.

`[ \Off ]`

Datový typ: `switch`

Robot se posune na naprogramovanou pozici pomocí nejbližší konfigurace osy.

Vykonávání programu

Jestliže je zvolen argument `\On` (nebo žádný argument), robot se posune na naprogramovanou pozici s konfiguračními parametry totožnými nebo blízkými daným konfiguračním parametrům.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je aktivní nahrazení programu nebo korekce dráhy, zvyšuje se riziko velkých pohybů, jelikož naprogramovaná konfigurační data jsou založena na původní pozici.

Jestliže je zvolen argument `\Off`, robot se vždy přesune k nejbližší konfiguraci osy. To může být odlišné k naprogramované skutečnosti, jestliže konfigurace byla nesprávně určena ručně nebo jestliže bylo provedeno nahrazení programu.

Kontrola konfigurace (`ConfJ \On`) je standardně aktivní s automatickým nastavením:

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Syntaxe

```
ConfJ
  [ '\' On] | [ '\' Off]';'
```

Související informace

Pro informace o	Viz
Zpracování různých konfigurací	*Technická referenční příručka - Přehled RAPID*
Konfigurace robotu během lineárního pohybu	*ConfJ - Kontroluje konfiguraci během lineárního pohybu na str 130*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.48 ConfJ - Kontroluje konfiguraci během lineárního pohybu

Použití

`ConfL` (*Configuration Linear*) se používá pro určení, jestli bude během lineárního nebo kruhového pohybu prováděno sledování konfigurace robotu. Jestliže není sledována, konfigurace v čase vykonávání se může lišit od naprogramované. Výsledkem mohou být také neočekávané široké pohyby robotu, když je měněn režim na pohyb spoje.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

POZNÁMKA

Pro IRB 5400 je sledování robotu vždy vypnuto, nezávisle na tom, co je stanoveno v `ConfL`.

Základní příklady

Následující příklady názorně ukazují instrukci `ConfL`:

Příklad 1

```
ConfL \On;
MoveL *, v1000, fine, tool1;
```

Vykonávání programu se zastaví, když naprogramované konfigurace není možné dosáhnout z aktuální pozice.

Příklad 2

```
SingArea \Wrist;
ConfL \On;
MoveL *, v1000, fine, tool1;
```

Robot se posune k naprogramované pozici, orientaci a konfiguraci osy zápěstí. Jestliže to není možné, vykonávání programu se zastaví.

Příklad 3

```
ConfL \Off;
MoveL *, v1000, fine, tool1;
```

Robot se posune k naprogramované pozici a orientaci, ale k nejbližší možné konfiguraci osy, což může být odlišné od naprogramované.

Argumenty

```
ConfL [\On]|[\Off]
```

`[ \On ]`

Datový typ: `switch`

Konfigurace robotu je sledována.

`[ \Off ]`

Datový typ: `switch`

Konfigurace robotu není sledována.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Během lineárního nebo kruhového pohybu se robot vždy pohybuje k naprogramované pozici a orientaci, která má nejbližší možnou konfiguraci osy. Jestliže je použit argument `\On` (nebo žádný argument), potom se vykonávání programu zastaví, jelikož je riziko, že konfigurace naprogramované pozice nebude možné dosáhnout z aktuální pozice. Způsob rozhodnutí se liší mezi typy robotů, viz *confdata - Konfigurační data robotu na str 1473*.

Před spuštěním přikázaného pohybu je provedena verifikace, aby bylo vidět, jestli je možné dosáhnout naprogramované pozice. Jestliže to není možné, program je zastaven. Po dokončení pohybu (v zóně nebo v jemném bodu) je také verifikováno, že robot dosáhl naprogramované pozice.

Jestliže se použije `SingArea\Wrist`, robot se vždy pohybuje k naprogramované konfiguraci osy zápěstí.

Jestliže je použit argument `\Off`, nebude zahrnuto sledování.

Po zastavení, které způsobila chyba konfigurace, je možné restartovat program RAPID v ručním režimu. Všimněte si, že v tomto případě, kvůli hlášené chybě, se robot pravděpodobně neposune ke správné konfiguraci.

Jednoduchým pravidlem pro předcházení problémům u `ConfL\On` a `\Off`, je vkládat mezilehlé body, aby pohyb každé osy byl menší než 180 stupňů mezi body.

Jestliže se použije `ConfL\Off` s velkým pohybem, může to způsobit zastavení přímo nebo později v programu s chybou `50050 Position outside reach` nebo `50080 Position not compatible`. V programu s `ConfL\Off` se doporučuje mít pohyby ke známým konfiguračním bodům s“`ConfJ\On` + `MoveJ`” nebo “`ConfL\On` + `SingArea\Wrist` + `MoveL`” jako startovní body pro různé části programu.

Sledování je aktivní jako standard. Je automaticky nastaveno:

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Syntaxe

```
ConfL
  [ '\' On] | [ '\' Off]';'
```

Související informace

Pro informace o	Viz
Zpracování různých konfigurací	*Technická referenční příručka - Přehled RAPID*
Konfigurace robotu během pohybu spoje	*ConfJ - Kontroluje konfiguraci během pohybu spoje na str 128*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.48 ConfJ - Kontroluje konfiguraci během lineárního pohybu

RobotWare - OS

Pokračování

Pro informace o	Viz
Definovat interpolaci kolem singulárních bodů	*SingArea - Definuje interpolaci kolem singulárních bodů na str 648*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*
Konfigurační data robotu	*confdata - Konfigurační data robotu na str 1473*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.49 CONNECT - Připojuje přerušení k trap rutině

Použití

`CONNECT` se používá pro nalezení identity přerušení a jeho připojení k trap rutině.

Přerušení je definováno přikázáním události přerušení a určením jeho identity. Když se událost objeví, provede se automaticky trap rutina.

Základní příklady

Následující příklad názorně ukazuje instrukci `CONNECT`:

Příklad 1

```
VAR intnum feeder_low;
PROC main()
  CONNECT feeder_low WITH feeder_empty;
  ISignalDI di1, 1 , feeder_low;
  ...
```

Identita přerušení `feeder_low` je vytvořena a připojena k trap rutině `feeder_empty`. Dojde k přerušení, když vstup `di1` se dostává vysoko. Jinými slovy, když tento signál je vysoký, trap rutina `feeder_empty` bude provedena.

Argumenty

```
CONNECT Interrupt WITH Trap routine
```

`Interrupt`

Datový typ: `intnum`

Proměnná, které má být přidělena identita přerušení. To nesmí být deklarováno v rámci rutiny (data rutiny).

`Trap routine`

Identifier

Jméno trap rutiny.

Vykonávání programu

Proměnné je přidělena identita přerušení, což by mělo být použito při přikazování nebo deaktivaci přerušení. Tato identita je také připojena k určené trap rutině.

POZNÁMKA

Všechna přerušení v úloze jsou zrušena, když je ukazatel programu nastaven na main pro tuto úlohu a musí být znovu připojen. Přerušení nebudou ovlivněna výpadkem napájení nebo **Restartem**.

Omezení

Přerušení (identita přerušení) nemůže být připojena k více než jedné trap rutině. Různá přerušení, nicméně, mohou být připojena ke stejné trap rutině.

Když bylo přerušení připojeno k trap rutině, nemůže být přepojeno nebo přeneseno k jiné rutině; nejprve musí být vymazáno pomocí instrukce `IDelete`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.49 CONNECT - Připojuje přerušení k trap rutině

RobotWare - OS

Pokračování

Přerušení, která nebyla řešena, když se vykonávání programu zastavilo, budou opomenuta. Přerušení nejsou brána v úvahu při zastavení programu. Přerušení, která byla nastavena jako bezpečná, nebudou opomenuta při zastavení programu. Budou řešena, až se program opět spustí.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_ALRDYCNT	Proměnná přerušení je již připojena k rutině TRAP.
ERR_CNTNOTVAR	Proměnná přerušení není referencí proměnné.
ERR_INOMAX	Nejsou už dostupná žádná čísla přerušení.

Syntaxe

```
CONNECT <connect target> WITH <trap>';'
<connect target> ::= <variable>
                   | <parameter>
                   | <VAR>
<trap> ::= <identifier>
```

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*
Více informací o správě přerušení	*Technická referenční příručka - Přehled RAPID*
Datový typ přerušení	*intnum - Identita přerušení na str 1516*
Zrušení přerušení	*IDelete - Zruší přerušení na str 243*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.50 CopyFile - Kopírovat soubor

Použití

`CopyFile` se používá k pořízení kopie existujícího souboru.

Základní příklady

Následující příklad názorně ukazuje instrukci `CopyFile`:

Příklad 1

CopyFile "HOME:/myfile", "HOME:/yourfile";

Soubor `myfile` je kopírován do `yourfile`. Oba soubory jsou potom identické.

```
CopyFile "HOME:/myfile", "HOME:/mydir/yourfile";
```

Soubor `myfile` se kopíruje do `yourfile` v adresáři `mydir`.

Argumenty

```
CopyFile OldPath NewPath
```

`OldPath`

Datový typ: `string`

Kompletní cesta souboru, ze kterého se bude kopírovat.

`NewPath`

Datový typ: `string`

Kompletní cesta souboru, do kterého se bude kopírovat.

Vykonávání programu

Soubor určený v `OldPath` bude zkopírován do souboru určeného v `NewPath`.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_FILEEXIST`	Soubor určený v `NewPath` už existuje.

Syntaxe

```
CopyFile
[ OldPath ':=' ] < expression (IN) of string > ','
[ NewPath ':=' ] < expression (IN) of string >';'
```

Související informace

Pro informace o	Viz
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.50 CopyFile - Kopírovat soubor

RobotWare - OS

Pokračování

Pro informace o	Viz
Zkontrolovat typ souboru	*IsFile - Zkontrolovat typ souboru na str 1207*
Zkontrolujte velikost souboru	*FileSize - Vyhledat velikost souboru na str 1155*
Zkontrolujte velikost souborového systému	*FSSize - Vyhledat velikost souborového systému na str 1161*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.51 CopyRawBytes - Kopírovat obsah dat rawbytes

Použití

`CopyRawBytes` se používá ke kopírování všech nebo části obsahů z jedné proměnné `rawbytes` do druhé.

Základní příklady

Následující příklad názorně ukazuje instrukci `CopyRawBytes`:

Příklad 1

```
VAR rawbytes from_raw_data;
VAR rawbytes to_raw_data;
VAR num integer := 8
VAR num float := 13.4;

ClearRawBytes from_raw_data;
PackRawBytes integer, from_raw_data, 1 \IntX := DINT;
PackRawBytes float, from_raw_data, (RawBytesLen(from_raw_data)+1)
     \Float4;
CopyRawBytes from_raw_data, 1, to_raw_data, 3,
     RawBytesLen(from_raw_data);
```

V tomto příkladu proměnná `from_raw_data` typu `rawbytes` je nejprve vyčištěna, potom jsou všechny byty nastaveny na 0. Potom je v prvních 4 bytech umístěna hodnota `integer` a v dalších 4 bytech hodnota `float`.

Po naplnění `from_raw_data` daty je obsah (8 bytů) kopírován do `to_raw_data`, se začátkem na pozici `3`.

Argumenty

```
CopyRawBytes FromRawData FromIndex ToRawData ToIndex[\NoOfBytes]
```

`FromRawData`

Datový typ: `rawbytes`

`FromRawData` je datový kontejner, ze kterého by měla být kopírována data `rawbytes`.

`FromIndex`

Datový typ: `num`

`FromIndex` je pozice v `FromRawData`, kde začínají data ke kopírování. Indexování začíná na 1.

`ToRawData`

Datový typ: `rawbytes`

`ToRawData` je datový kontejner, do kterého by měla být kopírována data `rawbytes`.

`ToIndex`

Datový typ: `num`

`ToIndex` je pozice v `ToRawData`, kam budou umístěna kopírovaná data. Zkopírováno je všechno až do konce. Indexování začíná na 1.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.51 CopyRawBytes - Kopírovat obsah dat rawbytes

RobotWare - OS

Pokračování

`[\NoOfBytes]`

Datový typ: `num`

Hodnota určená s `\NoOfBytes` **je počet bytů, které budou kopírovány z** `FromRawData` **do** `ToRawData`.

Jestliže není určeno `\NoOfBytes`**, všechny byty z** `FromIndex` **do konce aktuální délky platných bytů v** `FromRawData` **jsou zkopírovány.**

Vykonávání programu

Během vykonávání programu jsou kopírována data z jedné proměnné `rawbytes` **do druhé.**

Aktuální délka platných bytů v proměnné `ToRawData` **je nastavena na:**

- (`ToIndex` **+ copied_number_of_bytes - 1)**
- **Aktuální délka platných bytů v proměnné** `ToRawData` **se nezmění, jestliže kompletní kopírovací operace je provedena uvnitř staré aktuální délky platných bytů v proměnné** `ToRawData`.

Omezení

`CopyRawBytes` **se nemůže použít pro kopírování některých dat z jedné proměnné** `rawbytes` **do jiné části stejné proměnné** `rawbytes`.

Syntaxe

```
CopyRawBytes
  [FromRawData ':=' ] < variable (VAR) of rawbytes> ','
  [FromIndex ':=' ] < expression (IN) of num> ','
  [ToRawData ':=' ] < variable (VAR) of rawbytes> ','
  [ToIndex ':=' ] < expression (IN) of num>
  ['\'NoOfBytes ':=' < expression (IN) of num> ]';'
```

Související informace

Pro informace o	Viz
Data`rawbytes`	*rawbytes - Data raw na str 1559*
Získat délku dat `rawbytes`	*RawBytesLen - Získat délku dat rawbytes na str 1278*
Vyčistit obsah dat `rawbytes`	*ClearRawBytes - Vyčistit obsahy rawbyte dat na str 118*
Zabalit hlavičku DeviceNet do dat `rawbytes`	*PackDNHeader - Zabalit hlavičku DeviceNet do dat rawbytes na str 446*
Zabalit data do dat `rawbytes`	*PackRawBytes - Zabalit data do dat rawbytes na str 449*
Zapsat data `rawbytes`	*WriteRawBytes - Zapsat data rawbytes na str 994*
Načíst data `rawbytes`	*ReadRawBytes - Přečíst data rawbytes na str 530*
Rozbalit data z dat `rawbytes`	*UnpackRawBytes - Rozbalit data z rawbyte dat na str 908*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.52 CorrClear - Odstraní všechny generátory korekcí

Popisy

`CorrClear` se používá k odstranění všech připojených generátorů korekcí. Instrukci je možné používat k odstranění všech ofsetů poskytnutých dříve od všech generátorů korekcí.

Základní příklady

Následující příklad názorně ukazuje instrukci `CorrClear`:

Příklad 1

CorrClear;

Instrukce odstraňuje všechny připojené generátory korekcí

> **POZNÁMKA**
>
> Snadným způsobem, jak zajistit odstranění všech generátorů korekcí (s korekcemi) při startu programu, je vykonat `CorrClear` v událostní rutině `START`.
>
> Viz *Technická referenční příručka - Systémové parametry*, téma *Controller*.

Syntaxe

```
CorrClear ';'
```

Související informace

Pro informace o	Viz
Připojuje se ke generátoru korekcí	*CorrCon - Připojuje ke generátoru korekcí na str 140*
Odpojuje se od generátoru korekcí	*CorrDiscon - Odpojuje se od generátoru korekcí na str 145*
Zapisuje do generátoru korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
Načítá aktuální celkové ofsety	*CorrRead - Načítá aktuální celkové ofsety na str 1107*
Popisovač korekcí	*corrdescr - Popisovač generátoru korekcí na str 1480*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.53 CorrCon - Připojuje ke generátoru korekcí

Použití

`CorrCon` používá se pro připojení ke generátoru korekcí.

Základní příklady

Následující příklad názorně ukazuje instrukci `CorrCon`:

Viz také *Další příklady na str 140*.

Example1

```
VAR corrdescr id;
...
CorrCon id;
```

Reference generátoru korekcí odpovídá rezervaci proměnné `id`.

Argumenty

```
CorrCon Descr
```

`Descr`

Datový typ: `corrdescr`

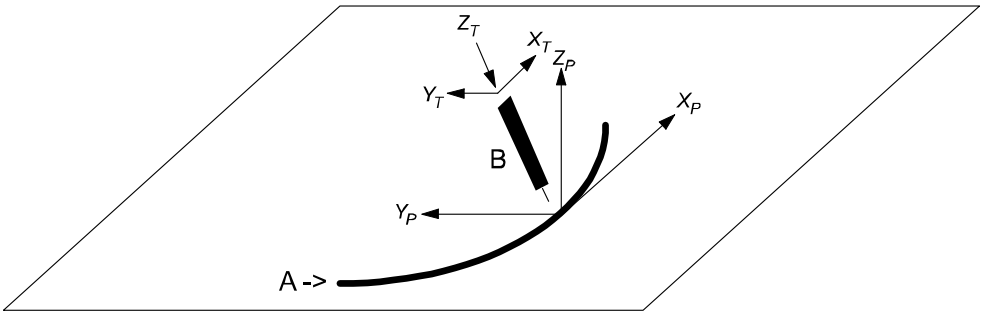
Popisovač generátoru korekcí.

Další příklady

Více příkladů instrukce `CorrCon` je názorně uvedeno dole.

souřadnicový systém dráhy

Všechny korekce dráhy (ofsety na dráze) se doplňují do souřadnicového systému dráhy. Souřadnicový systém dráhy je definován, jak je uvedeno dole:



xx0500002156

A	Směr dráhy
B	Nástroj
X	souřadnicový systém dráhy
X	Souřadnicový systém nástroje

- Osa X souřadnice dráhy je dána jako tečna dráhy.
- Osa Y souřadnice dráhy je odvozena jako vektorový součin osy X souřadnice dráhy a osy Z souřadnice nástroje.

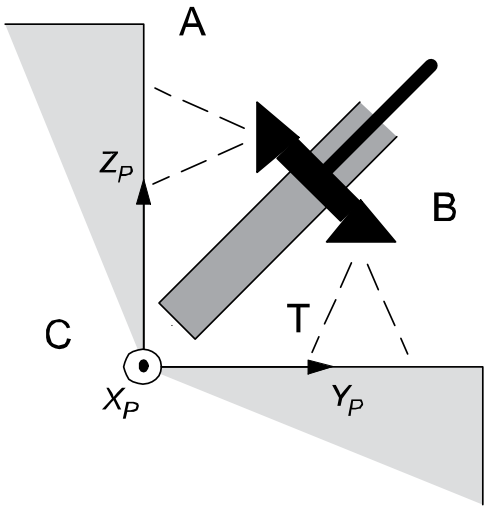
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

- Osa Z souřadnice dráhy je odvozena jako vektorový součin osy X souřadnice dráhy a osy Y souřadnice dráhy.

Příklad aplikace

Příkladem aplikace používající korekce dráhy je robot držící nástroj s dvěma senzory namontovanými na něm kvůli detekci svislých a vodorovných vzdáleností k pracovnímu objektu. Obrázek dole ilustruje zařízení na korekci dráhy.



xx0500002155

A	Senzor pro vodorovnou korekci
B	Senzor pro svislou korekci
C	souřadnicový systém dráhy
T	Nástroj

Příklad programu

POZNÁMKA

`hori_sig` a `vert_sig` jsou analogové signály definované v parametrech systému.

```
CONST num TARGET_DIST := 5;
CONST num SCALE_FACTOR := 0.5;
VAR intnum intno1;
VAR corrdescr hori_id;
VAR corrdescr vert_id;
VAR pos total_offset;
VAR pos write_offset;
VAR bool conFlag;
PROC PathRoutine()
  ! Connect to the correction generators for horizontal and vertical
       correction.
  CorrCon hori_id;
  CorrCon vert_id;
  conFlag := TRUE;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  ! Setup a 5 Hz timer interrupt. The trap routine will read the
       sensor values and
  ! compute the path corrections.
  CONNECT intno1 WITH ReadSensors;
  ITimer\Single, 0.2, intno1;

  ! Position for start of contour tracking
  MoveJ p10, v100, z10, tool1;
  ! Run MoveL with both vertical and horizontal correction.
  MoveL p20, v100, z10, tool1 \Corr;

  ! Read the total corrections added by all connected correction
       generators.
  total_offset := CorrRead();
  ! Write the total vertical correction on the FlexPendant.
  TPWrite "The total vertical correction is:" \Num:=total_offset.z;

  ! Disconnect the correction generator for vertical correction.
  ! Horizontal corrections will be unaffected.
  CorrDiscon vert_id;
  conFlag := FALSE;

  ! Run MoveL with only horizontal interrupt correction.
  MoveL p30, v100, fine, tool1 \Corr;
  ! Remove all outstanding connected correction generators.
  ! In this case, the only connected correction generator is the
       one for horizontal
  ! correction.
  CorrClear;
  ! Remove the timer interrupt.
  IDelete intno1;
ENDPROC
TRAP ReadSensors
  VAR num horiSig;
  VAR num vertSig;

  ! Compute the horizontal correction values and execute the
       correction.
  horiSig := hori_sig;
  write_offset.x := 0;
  write_offset.y := (hori_sig - TARGET_DIST)*SCALE_FACTOR;
  write_offset.z := 0;
  CorrWrite hori_id, write_offset;

  IF conFlag THEN
    ! Compute the vertical correction values and execute the
         correction.
    write_offset.x := 0;
    write_offset.y := 0;
    write_offset.z := (vert_sig - TARGET_DIST)*SCALE_FACTOR;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
    CorrWrite vert_id, write_offset;
  ENDIF
  !Setup interrupt again
  IDelete intnol;
  CONNECT intno1 WITH ReadSensors;
  ITimer\single, 0.2, intno1;
ENDTRAP
```

Vysvětlení programu

Dva generátory korekcí jsou připojeny s instrukcí `CorrCon`. Každý generátor korekcí je odkazován jedinečným popisovačem (`hori_id` a `vert_id`) typu `corrdescr`. Každý ze dvou senzorů bude používat jeden generátor korekcí.

Přerušení časovače je nastaveno na volání trap rutiny `ReadSensors` s kmitočtem 5 Hz. Ofsety potřebné korekci dráhy jsou vypočítány v trap rutině a zapsány do odpovídajícího generátoru korekcí (odkazovaného popisovači) `hori_id` a `vert_id`) instrukcí `CorrWrite`. Všechny korekce budou mít okamžitý efekt na dráhu.

Instrukce `MoveL` musí být naprogramována s přepínacím argumentem `Corr`, když se používají korekce dráhy. Jinak nebudou provedeny žádné korekce.

Když je první instrukce `MoveL` připravena, funkce `CorrRead` se použije pro načtení součtu všech korekcí (celková korekce dráhy) daných všemi připojeným generátory korekcí. Výsledek celkové korekce svislé dráhy je zapsán do FlexPendantu s instrukcí `TPWrite`.

`CorrDiscon` potom odpojí generátor korekcí pro svislou korekci (odkazovanou popisovačem `vert_id`). Všechny korekce přidané tímto generátorem korekcí budou odstraněny z celkové korekce dráhy. Korekce přidané generátorem korekcí pro vodorovnou korekci budou stále zachovány.

Nakonec, funkce `CorrClear` odstraní všechny zbylé připojené generátory korekcí a jejich dříve přidané korekce. V tomto případě je to pouze generátor korekcí pro vodorovnou korekci, který bude odstraněn. Přerušení časovače bude také odstraněno instrukcí `IDelete`.

Generátory korekcí

Tabulka dole ilustruje generátory korekcí.

X	Y	Z	Osa souřadnice dráhy
0	0	3	Generátor svislé korekce se součtem všech svých korekcí dráhy
0	1	0	Generátor vodorovné korekce se součtem všech svých korekcí dráhy
-	-	-	Nepřipojený generátor korekcí
-	-	-	Nepřipojený generátor korekcí
-	-	-	Nepřipojený generátor korekcí
0	1	3	Součet všech korekcí provedených všemi připojenými generátory korekcí

Omezení

Současně může být připojeno max. 5 generátorů korekcí.

Připojené generátory korekcí nepřežijí restart řadiče.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.53 CorrCon - Připojuje ke generátoru korekcí

Path Offset

Pokračování

Syntaxe

```
CorrCon
  [ Descr ':=' ] < variable (VAR) of corrdescr > ';'
```

Související informace

Pro informace o	Viz
Odpojuje se od generátoru korekcí	*CorrDiscon - Odpojuje se od generátoru korekcí na str 145*
Zapisuje do generátoru korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
Načítá aktuální celkové ofsety	*CorrRead - Načítá aktuální celkové ofsety na str 1107*
CorrClear - Odstraní všechny generátory korekcí	*CorrClear - Odstraní všechny generátory korekcí na str 139*
Popisovač generátoru korekcí	*corrdescr - Popisovač generátoru korekcí na str 1480*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.54 CorrDiscon - Odpojuje se od generátoru korekcí

Popis

`CorrDiscon` se používá pro odpojení od dříve připojeného generátoru korekcí. Instrukci je možné používat k odstranění korekcí zadaných dříve.

Základní příklady

Následující příklad názorně ukazuje instrukci `CorrDiscon`:

Viz také *Další příklady na str 145*.

Příklad 1

```
VAR corrdescr id;
...
CorrCon id;
...
CorrDiscon id;
```

`CorrDiscon` odpojuje od dříve připojeného generátoru korekcí odkazovaného id popisovače.

Argumenty

```
CorrDiscon Descr
```

`Descr`

Datový typ: `corrdescr`

Popisovač generátoru korekcí.

Další příklady

Více informací o instrukci CorrDiscon, viz *CorrCon - Připojuje ke generátoru korekcí na str 140*.

Syntaxe

```
CorrDiscon
  [ Descr ':=' ] < variable (VAR) of corrdescr > ';'
```

Související informace

Pro informace o	Viz
Připojuje se ke generátoru korekcí	*CorrCon - Připojuje ke generátoru korekcí na str 140*
Zapisuje do generátoru korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
Načítá aktuální celkové ofsety	*CorrRead - Načítá aktuální celkové ofsety na str 1107*
CorrClear - Odstraní všechny generátory korekcí	*CorrClear - Odstraní všechny generátory korekcí na str 139*
Popisovač korekcí	*corrdescr - Popisovač generátoru korekcí na str 1480*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.55 CorrWrite - Zapisuje do generátoru korekcí

Popis

`CorrWrite` se používá pro zápis ofsetů v souřadnicovém systému dráhy do generátoru korekcí.

Základní příklady

Následující příklad názorně ukazuje instrukci `CorrWrite`:

Příklad 1

```
VAR corrdescr id;
VAR pos offset;
...
CorrWrite id, offset;
```

Aktuální ofsety uložené v ofsetu proměnné jsou zapsány do generátoru korekcí odkazovaného id popisovače.

Argumenty

```
CorrWrite Descr Data
```

Popis

Datový typ: `corrdescr`

Popisovač generátoru korekcí.

Data

Datový typ: `pos`

Ofset, který bude zapsán.

Další příklady

Více příkladů instrukce CorrWrite: viz *CorrCon - Připojuje ke generátoru korekcí na str 140*.

Omezení

Nejlepšího provedení se dosahuje na přímých drahách. Jelikož rychlost a úhly mezi následujícími lineárními dráhami narůstají, odchylka od očekávané dráhy bude také narůstat. Stejné se vztahuje na kruhy s narůstajícím poloměrem kruhu.

Syntaxe

```
CorrWrite
  [ Descr ':=' ] < variable (VAR) of corrdescr > ','
  [ Data ':=' ] < expression (IN) of pos > ';'
```

Související informace

Pro informace o	Viz
Připojuje se ke generátoru korekcí	*CorrCon - Připojuje ke generátoru korekcí na str 140*
Odpojuje se od generátoru korekcí	*CorrDiscon - Odpojuje se od generátoru korekcí na str 145*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Načítá aktuální celkové ofsety	*CorrRead - Načítá aktuální celkové ofsety na str 1107*
CorrClear - Odstraní všechny generátory korekcí	*CorrClear - Odstraní všechny generátory korekcí na str 139*
Popisovač generátoru korekcí	*corrdescr - Popisovač generátoru korekcí na str 1480*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.56 DeactEventBuffer - Deaktivace zásobníku událostí

Popis

`DeactEventBuffer` se používá pro deaktivaci použití zásobníku událostí v programové úloze aktuálního pohybu.

Instrukce `DeactEventBuffer` a `ActEventBuffer` by se měly používat při kombinaci aplikace pomocí jemných bodů a pokračující aplikace, kde je nutné nastavovat signály předem kvůli pomalému procesnímu vybavení.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `DeactEventBuffer`:

Příklad 1

```
..
DeactEventBuffer;
! Use an application that use finepoints, such as SpotWelding
..
! Activate the event buffer again
ActEventBuffer;
! Now it is possible to use an application that needs
! to set signals in advance, such as Dispense
..
```

`DeactEventBuffer` deaktivuje konfigurovaný zásobník událostí. Při používání aplikace s jemnými body bude start robotu od jemného bodu rychlejší. Při aktivaci zásobníku událostí s `ActEventBuffer` je možné nastavovat signály dopředu pro aplikaci s pomalým procesním vybavením.

Vykonávání programu

Deaktivace zásobníku událostí se vztahuje k dalšímu provedenému pohybu robotu a je platné až do provedení instrukce `ActEventBuffer`.

Instrukce bude před aktivací zásobníku událostí čekat, až robot a externí osy dosáhnou stop bodu (`ToPoint` aktuální instrukce pohybu). Proto se doporučuje programovat pohybovou instrukci předcházející `DeactEventBuffer` s jemným bodem.

Výchozí hodnota (použijte zásobník událostí = TRUE) se nastavuje automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

`DeactEventBuffer` se nemůže provádět v RAPID rutině připojené k žádné z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart nebo Step.

Syntaxe

```
DeactEventBuffer ';'
```

Související informace

Pro informace o	Viz
Deaktivace zásobníku událostí	*ActEventBuffer - Aktivace vyrovnávací paměti pro události na str 22*
Konfigurace Event preset time	*Technická referenční příručka - Systémové parametry*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.57 DeactUnit - Deaktivuje mechanickou jednotku

Použití

`DeactUnit` se používá pro deaktivaci mechanické jednotky.

Může se používat pro určení, která jednotka bude aktivní, když se používají například společné pohonné jednotky.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Příklady

Následující příklady názorně ukazují instrukci `DeactUnit`:

Příklad 1

```
DeactUnit orbit_a;
```

Deaktivace mechanické jednotky `orbit_a`.

Příklad 2

```
MoveL p10, v100, fine, tool1;
DeactUnit track_motion;
MoveL p20, v100, z10, tool1;
MoveL p30, v100, fine, tool1;
ActUnit track_motion;
MoveL p40, v100, z10, tool1;
```

Jednotka `track_motion` bude stacionární, když se robot posune k `p20` a `p30`. Potom se robot a `track_motion` posune k `p40`.

Příklad 3

```
MoveL p10, v100, fine, tool1;
DeactUnit orbit1;
ActUnit orbit2;
MoveL p20, v100, z10, tool1;
```

Jednotka `orbit1` je deaktivována a `orbit2` je aktivována.

Argumenty

```
DeactUnit MechUnit
```

MechUnit

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky, která má být deaktivována.

Vykonávání programu

Když je připravena skutečná dráha robotu a externích os, dráha na úrovni aktuální dráhy je vyčištěna a určená mechanická jednotka je deaktivována. To znamená, že nebude ani kontrolována ani sledována, dokud nebude znovu aktivována.

Jestliže několik mechanických jednotek sdílí společnou pohonnou jednotku, deaktivace jedné z těchto mechanických jednotek odpojí rovněž tuto jednotku od společné pohonné jednotky.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Instrukce `DeactUnit` se nemůže používat, když jedna z mechanických jednotek je v nezávislém režimu.

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (zonedata `fine`), nikoliv s průjezdným bodem, jinak nebude možný restart po selhání napájení.

`DeactUnit` se nemůže provádět v RAPID rutině připojené ke kterékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart nebo Step.

Je možné používat `ActUnit` - `DeactUnit` na úrovni `StorePath`, ale stejné mechanické jednotky musí být aktivní při provádění `RestoPath` jako kdyby bylo provedeno `StorePath`. Při takové operace budou záznamník dráhy a dráha na základní úrovni nedotčeny, ale dráha na úrovni `StorePath` bude vyčištěna.

Syntaxe

```
DeactUnit
  [MechUnit ':='] < variable (VAR) of mecunit> ';'
```

Související informace

Pro informace o	Viz
Aktivace mechanických jednotek	*ActUnit - Aktivuje mechanickou jednotku na str 24*
Mechanické jednotky	*mecunit - Mechanická jednotka na str 1531*
Zkontrolujte, jestli je mechanická jednotka aktivována nebo nikoliv.	*IsMechUnitActive - Je mechanická jednotka aktivní na str 1211*
Záznamník dráhy	*PathRecMoveBwd - Posunout záznamník dráhy dozadu na str 458mecunit - Mechanická jednotka na str 1531*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.58 Decr - Snižování po 1

Použití

`Decr` se používá pro odečtení 1 od numerické proměnné nebo perzistentu.

Základní příklady

Následující příklad názorně ukazuje instrukci `Decr`:

Viz také *Další příklady na str 152*.

Příklad 1

```
Decr reg1;
```

`1` je odečteno od `reg1`, tj. `reg1:=reg1-1`.

Argumenty

```
Decr Name | Dname
```

`Name`

Datový typ: `num`

Jméno proměnné nebo perzistentu, které budou sníženy.

`Dname`

Datový typ: `dnum`

Jméno proměnné nebo perzistentu, které budou sníženy.

Další příklady

Více příkladů instrukce `Decr` je názorně uvedeno dole.

Příklad 1

```
VAR num no_of_parts:=0;
...
TPReadNum no_of_parts, "How many parts should be produced? ";
WHILE no_of_parts>0 DO
  produce_part;
  Decr no_of_parts;
ENDWHILE
```

Operátor je požádán o vložení počtu kusů, které je třeba vyrobit. Používá se proměnná `no_of_parts` pro spočítání počtu, který je třeba ještě vyrobit.

Příklad 2

```
VAR dnum no_of_parts:=0;
...
TPReadDnum no_of_parts, "How many parts should be produced? ";
WHILE no_of_parts>0 DO
  produce_part;
  Decr no_of_parts;
ENDWHILE
```

Operátor je požádán o vložení počtu kusů, které je třeba vyrobit. Používá se proměnná `no_of_parts` pro spočítání počtu, který je třeba ještě vyrobit.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
Decr
  [ Name ':=' ] < var or pers (INOUT) of num >
  | [ Dname ':=' ] < var or pers (INOUT) of dnum >' ;'
```

Související informace

Pro informace o	Viz
Přírůstkování proměnné o 1	*Incr - Přírůstky po 1 na str 251*
Odečtení jakékoliv hodnoty od proměnné	*Add - Přidává numerickou hodnotu na str 26*
Změna dat pomocí libovolného výrazu, například násobení	*":=" - Přiděluje hodnotu na str 33*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.59 DitherAct - Zapíná volnoběh pro soft servo

Použití

`DitherAct` se používá pro zapnutí funkce volnoběh, která omezí chvění v soft servu u IRB 7600.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `DitherAct`:

Příklad 1

```
SoftAct \MechUnit:=ROB_1, 2, 100;
WaitTime 2;
DitherAct \MechUnit:=ROB_1, 2;
WaitTime 1;
DitherDeact;
SoftDeact;
```

Volnoběh se zapíná v soft servu pouze na jednu sekundu.

Příklad 2

```
DitherAct \MechUnit:=ROB_1, 2;
SoftAct \MechUnit:=ROB_1, 2, 100;
WaitTime 1;
MoveL p1, v50, z20, tool1;
SoftDeact;
DitherDeact;
```

Volnoběh je zapnut pro osu 2. Pohyb se opozdí o jednu sekundu, aby byl umožněn dostatečný přechodový čas pro rampu `SoftAct`. Jestliže `DitherAct` je volán před `SoftAct`, volnoběh bude spuštěn, kdykoliv je `SoftAct` vykonáván pro tuto osu. Jestliže není volán žádný `DitherDeact`, volnoběh zůstane zapnut pro všechna následující volání `SoftAct`.

Argumenty

```
DitherAct [\MechUnit] Axis [\Level]
```

`[ \MechUnit ]`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky. Jestliže je argument vynechán, znamená to aktivaci soft serva pro určené osy robotu.

`Axis`

Datový typ: `num`

Číslo osy (1-6).

`[ \Level ]`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Amplituda volnoběhu (50-150 %). Při 50 % jsou oscilace omezeny (zvýšené tření). Při 150 % je amplituda maximální (výsledkem mohou být vibrace koncového efektoru). Výchozí hodnota je 100 %.

Vykonávání programu

`DitherAct` může být volán před nebo po `SoftAct`. Volání `DitherAct` po `SoftAct` je rychlejší, ale má jiná omezení.

Volnoběh se obvykle nevyžaduje pro osu 1 modelu IRB 7600. Nejvyšší efekt omezení tření je na osách 2 a 3.

Parametry volnoběhu jsou samonastavitelné. Plného výkonu volnoběhu se dosáhne po třech nebo čtyřech provedeních `SoftAct` v procesní pozici.

Omezení

Volání `DitherAct` po `SoftAct` může způsobit nechtěný pohyb robotu. Jediným způsobem, jak eliminovat toto chování, je volání `DitherAct` před `SoftAct`. Jestliže pohyb zůstává, čas rampy `SoftAct` by měl být prodloužen.

Přechodový čas je čas rampy, který se liší mezi roboty, násobený faktorem rampy instrukce `SoftAct`.

Volnoběh není dostupný pro osu 6.

Volnoběh se vždy deaktivuje při výpadku napájení.

Instrukci je možné používat pouze u IRB 7600.

VAROVÁNÍ

Při volání `DitherAct` před `SoftAct` robot **musí být v jemném bodu**. Také, **opuštění jemného bodu není dovoleno**, dokud neuplyne přechodový čas rampy. **Toto může poškodit převodové skříně**.

Syntaxe

```
DitherAct
  [ '\' MechUnit ':=' < variable (VAR) of mecunit > ]
  [Axis ':=' ] < expression (IN) of num >
  [ '\' Level ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Aktivace sof serva	*SoftAct - Aktivace měkkého serva na str 685*
Chování se zařazeným soft servem	*Technická referenční příručka - Přehled RAPID*
Vypnout volnoběh	*DitherDeact - Vypíná volnoběh pro soft servo na str 156*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.60 DitherDeact - Vypíná volnoběh pro soft servo

Použití

`DitherDeact` se používá pro vypnutí funkce volnoběhu pro soft servo IRB 7600.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `DitherDeact`:

Příklad 1

```
DitherDeact;
```

Deaktivuje volnoběh na všech osách.

Vykonávání programu

`DitherDeact` se může používat kdykoliv. V soft servo volnoběh zastaví okamžitě na všech osách. Mimo soft servo nebude volnoběh aktivní, když je vykonán další `SoftAct`.

Volnoběh je automaticky vypnut

- při **Restartu**.
- když je nový program načten.
- při spuštění provedení programu od začátku.

Syntaxe

```
DitherDeact';'
```

Související informace

Pro informace o	Viz
Aktivace volnoběhu	*DitherAct - Zapíná volnoběh pro soft servo na str 154*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.61 DropSensor - Shodit objekt nebo senzor

Použití

`DropSensor` se používá k odpojení od aktuálního objektu a program je připraven na další.

`DropSensor` se používá pro synchronizaci senzoru, ale nikoliv pro analogovou synchronizaci.

Základní příklad

```
MoveL *, v1000, z10, tool, \WObj:=wobj0;
SyncToSensor Ssync1\Off;
MoveL *, v1000, fine, tool, \WObj:=wobj0;
DropSensor Ssync1;
MoveL *, v1000, z10, tool, \WObj:=wobj0;
```

Argumenty

```
DropSensor MechUnit
```

`MechUnit`

Mechanical Unit

Datový typ: `mecunit`

Pohybující se mechanická jednotka, ke které se vztahuje pozice robotu v instrukci.

Vykonávání programu

Shození objektu znamená, že jednotka kodéru už nebude dále objekt sledovat. Objekt je odstraněn z objektové fronty a nemůže být obnoven.

Omezení

Jestliže je instrukce vydána, zatímco robot aktivně používá objekt senzoru,, potom se pohyb zastaví. Instrukce musí být vydána po průchodu robotu posledním synchronizovaným `robtarget`.

Instrukci je možné vydat pouze po použitém nesynchronizovaném pohybu v předchozích pohybových instrukcích buď s jemným bodem nebo několika (>1) rohovými zónami.

Syntaxe

```
DropSensor
  [ MechUnit ':='] < variable (VAR) of mecunit> ';'
```

Související informace

Pro informace o	Viz
Čekejte na připojení na senzor	*WaitSensor - Čekat na připojení na senzor na str 953*
Sync k senzoru	*SyncToSensor - Synch k senzoru na str 770*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.62 DropWObj - Shodit pracovní objekt na dopravník

Použití

`DropWObj` (*Drop Work Object*) se používá k odpojení od aktuálního objektu a program je připraven na další objekt na dopravníku.

Základní příklady

Následující příklad názorně ukazuje instrukci `DropWObj`:

Příklad 1

```
MoveL *, v1000, z10, tool, \WObj:=wobj_on_cnv1;
MoveL *, v1000, fine, tool, \WObj:=wobj0;
DropWObj wobj_on_cnv1;
MoveL *, v1000, z10, tool, \WObj:=wobj0;
```

Argumenty

```
DropWObj WObj
```

`WObj`

Work Object

Datový typ: `wobjdata`

Pohyblivý pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci. Dopravník mechanické jednotky bude určen od `ufmec` v pracovním objektu.

Vykonávání programu

Shození pracovního objektu znamená, že jednotka kodéru už nebude dále objekt sledovat. Objekt je odstraněn z objektové fronty a nemůže být obnoven.

Omezení

Jestliže instrukce vydána, zatímco robot aktivně používá pracovní objekt koordinovaný dopravníkem, potom se pohyb zastaví.

Instrukci je možné vydat pouze po použití pevného pracovního objektu v předchozích pohybových instrukcích buď s jemným bodem nebo několika (>1) rohovými zónami.

Syntaxe

```
DropWObj
  [ WObj ':='] < persistent (PERS) of wobjdata>';'
```

Související informace

Pro informace o	Viz
Čekejte na pracovní objekty	*WaitWObj - Čekejte na pracovní objekt nebo dopravník na str 972*
Sledování dopravníku	*Application manual - Sledování dopravníku*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.63 EGMActJoint - Připravit pohyb EGM pro cíl spoje

Použití

`EGMActJoint` aktivuje určitý proces EGM a definuje statická data pro pohyb vedený senzorem k cíli spoje, tj. datům, která nejsou často měněna mezi různými pohyby EGM.

Základní příklady

```
VAR egmident egmID1;
PERS pose pose1:=[[0,0,0], [1,0,0,0]];
CONST egm_minmax egm_minmax1:=[-1,1];

EGMGetId egmID1;
EGMSetupAI ROB_1, egmID1, "default" \Pose \aiR1x:=ai_01
     \aiR2y:=ai_02 \aiR3z:=ai_03 \aiR4rx:=ai_04 \aiR5ry:=ai_05
     \aiR6rz:=ai_06;
EGMActJoint egmID1 \J1:=egm_minmax1 \J3:=egm_minmax1
     \J4:=egm_minmax1;
```

Argumenty

```
EGMActJoint EGMid [\Tool] [\WObj] [\TLoad] [\J1] [\J2] [\J3] [\J4]
     [\J5] [\J6] [\LpFilter] [\SampleRate] [\MaxPosDeviation]
     [\MaxSpeedDeviation]
```

`EGMid`

Datový typ: `egmident`

identita EGM.

`[\Tool]`

Datový typ: `tooldata`

Nástroj použitý pro pohyby prováděné s instrukcí `EGMRunJoint`.

Argument `[\Tool]` je volitelný. Výchozí hodnota, když je argument vypuštěn, je `tool0`.

`[\Wobj]`

Datový typ: `wobjdata`

Pracovní objekt použitý pro pohyby prováděné s instrukcí `EGMRunJoint`.

Argument `[\Wobj]` je volitelný. Výchozí hodnota, když je argument vypuštěn, je `wobj0`.

`[\TLoad]`

Total load

Datový typ: `loaddata`

Zatížení použité pro pohyby prováděné s instrukcí `EGMRunJoint`.

Argument `[ \TLoad ]` je volitelný. Výchozí hodnota, když je argument vypuštěn, je `load0`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.63 EGMActJoint - Připravit pohyb EGM pro cíl spoje
Externally Guided Motion
Pokračování

Argument \TLoad popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument \TLoad, potom loaddata v aktuálním tooldata není uvažován.

Když je argument \TLoad nastaven na load0, potom není argument \TLoad uvažován a loaddata v aktuálním tooldata je použit místo toho.

Aby bylo možné použít argument \TLoad, je nezbytné nastavit hodnotu systémového parametru ModalPayLoadMode na 0. Jestliže ModalPayLoadMode je nastaven na 0, není dále možné používat instrukci GripLoad.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr ModalPayLoadMode nastaven na 0, operátor má možnost kopírovat loaddata z nástroje do existující nebo nové proměnné perzistentu loaddata při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému SimMode (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, loaddata ve volitelném argumentu \TLoad není uvažován a loaddata v aktuálním tooldata se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci GripLoad. Proto je výchozí hodnota systémového parametru ModalPayLoadMode 1.

[\J1] [\J2] [\J3] [\J4] [\J5] [\J6]

Datový typ: egm_minmax

Konvergenční kritérium pro spoj 1 až 6 ve stupních. Výchozí hodnota je ±0,5 stupně.

Data konvergenčního kritéria se používají pro rozhodnutí, jestli robot dosáhl přikázaných pozic spojů. Jestliže rozdíl mezi přikázanou pozicí spoje a skutečnou pozicí spoje je v rozmezí egm_minmax.min a egm_minmax.max, má se zato, že spoj dosáhl své přikázané pozice. Když nejsou určena žádná konvergenční kritéria pro spoj, který byl zvolen v EGMRunJoint, použije se výchozí hodnota.

Jakmile všechny spoje, které byly určeny v EGMRunJoint, dosáhnou svých přikázaných pozic, samotný robot dosáhl své přikázané pozice a vykonávání RAPID pokračuje s další instrukcí RAPID.

[\LpFilter]

Datový typ: num

Šířka dolní propusti, v Hertzích (Hz), používá se pro filtrování šumu senzoru.

[\SampleRate]

Datový typ: num

Vzorkovací kmitočet načítání vstupních dat v násobcích 4 milisekund. Platné hodnoty jsou 4, 8, 12, 16, atd.

Výchozí hodnota je 4 milisekundy.

[\MaxPosDeviation]

Datový typ: num

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Max. odchylka spoje od naprogramované pozice ve stupních, tj. jemný bod, na kterém začal pohyb EGM. Stejná hodnota se používá pro všechny spoje.

Výchozí hodnota je 1000 stupňů.

`[\MaxSpeedDeviation]`

Datový typ: `num`

Max. přijatelná změna rychlosti spoje ve stupních/sekundách, tzn. že toto je faktor, pomocí kterého se upraví zrychlení/zpomalení.

Výchozí hodnota je 1,0 stupňů/sekund.

Omezení

- Jestliže je několikrát proveden `EGMActJoint` se stejným `EGMid`, poslední aktivační data se použijí pro instrukce `EGMRunJoint`, které následují, dokud běží nový `EGMActJoint`.
- `EGMActJoint` může být použit pouze v pohybových úlohách RAPID.

Syntaxe

```
EGMActJoint
  [EGMid ':='] <variable (VAR) of egmident>
  ['\'Tool ':=' <persistent (PERS) of tooldata>]
  ['\'Wobj ':=' <persistent (PERS) of wobjdata>
  ['\'TLoad ':=' <persistent (PERS) of loaddata>]
  ['\'J1 ':=' <expression (IN) of egm_minmax>]
  ['\'J2 ':=' <expression (IN) of egm_minmax>]
  ['\'J3 ':=' <expression (IN) of egm_minmax>]
  ['\'J4 ':=' <expression (IN) of egm_minmax>]
  ['\'J5 ':=' <expression (IN) of egm_minmax>]
  ['\'J6 ':=' <expression (IN) of egm_minmax>]
  ['\'LpFilter ':=' <expression (IN) of num>]
  ['\'SampleRate ':=' <expression (IN) of num>]
  ['\'MaxPosDeviation ':=' <expression (IN) of num>]
  ['\'MaxSpeedDeviation ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*
Datový typ egm_minmax	*egm_minmax - Konvergenční kritérium pro EGM na str 1490*
Instrukce EGMRunJoint	*EGMRunJoint - Provést pohyb EGM s cílem spoje na str 176*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.64 EGMActMove - Připravit pohyb EGM s korekcí dráhy

Použití

`EGMActMove` se používá pro aktivaci konkrétního procesu EGM a definuje statická data pro pohyb s korekcí dráhy, tzn. data, která nejsou často měněna mezi různými pohyby korekce dráhy EGM.

Základní příklady

Následující příklad názorně ukazuje instrukci `EGMActMove`.

Příklad 1

```
VAR egmident EGMid1;
PERS tooldata tLaser := [TRUE, [[148,50,326],
     [0.3902618,-0.589657,-0.589656,0.3902630]],
     [1,[-0.92,0,-0.39], [1,0,0,0], 0,0,0]];
EGMGetId EGMid1;
EGMSetupLTAPP ROB_1, EGMid1, "pathCorr", "OptSim", 1\LATR;
EGMActMove EGMid1, tLaser.tframe\SampleRate:=48;
```

Tento program registruje proces EGM a nastavuje senzor, který používá komunikační protokol LTAPP a je typu *look-ahead* (výhled) jako datový zdroj (senzor). Senzor by měl používat pro sledování číslo definice 1 typu spoje. Rychlost, kterou řadič přistupuje k zařízení a rámec senzoru zařízení jsou také nastaveny.

Argumenty

```
EGMActMove EGMid, SensorFrame [\SampleRate]
```

`EGMid`

Datový typ: `egmident`

Identita EGM.

`SensorFrame`

Datový typ: `pose`

Rámec senzoru.

`[\SampleRate]`

Datový typ: `num`

Vzorkovací kmitočet načítání vstupních dat v násobcích 24 milisekund. Platné hodnoty jsou 24, 48, 72, atd.

Vykonávání programu

Rámec senzoru a vzorkovací kmitočet senzoru jsou připojeny k identitě EGM, dokud nejsou buď resetovány s `EGMReset` nebo změněny jinou instrukcí `EGMActMove`.

Syntaxe

```
EGMActMove
  [EGMid ':='] <variable (VAR) of egmident> ','
  [SensorFrame ':='] < expression (IN) of pose>
  ['\'SampleRate ':=' <expression (IN) of num>] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.65 EGMActPose - Připravit pohyb EGM pro poziční cíl

Použití

`EGMActPose` aktivuje určitý proces EGM a definuje statická data pro pohyb vedený senzorem k pozičnímu cíli, tj. datům, která nejsou často měněna mezi různými pohyby EGM.

Základní příklady

```
VAR egmident egmID1;
PERS pose pose1:=[[0,0,0], [1,0,0,0]];
CONST egm_minmax egm_minmax_lin:=[-0.1,0.1];
CONST egm_minmax egm_minmax_rot:=[-0.1,0.2];
CONST pose posecor:=[[1200,400,900], [0,0,1,0]];
CONST pose posesens:=[[12.3313,-0.108707,416.142],
     [0.903899,-0.00320735,0.427666,0.00765917]];

EGMGetId egmID1;
EGMSetupAI ROB_1, egmID1, "default" \Pose \aiR1x:=ai_01
     \aiR2y:=ai_02 \aiR3z:=ai_03 \aiR4rx:=ai_04 \aiR5ry:=ai_05
     \aiR6rz:=ai_06;
EGMActPose egmID1 \Tool:=tool0 \WObj:=wobj0, posecor,
     EGM_FRAME_WOBJ, posesens, EGM_FRAME_TOOL \x:=egm_minmax_lin
     \y:=egm_minmax_lin \z:=egm_minmax_lin \rx:=egm_minmax_rot
     \ry:=egm_minmax_rot \rz:=egm_minmax_rot \LpFilter:=20;
```

Argumenty

```
EGMActPose EGMid [\Tool] [\WObj] [\TLoad], CorrFrame, CorrFrType,
     SensorFrame, SensorFrType [\x] [\y] [\z] [\rx] [\ry] [\rz]
     [\LpFilter] [\SampleRate] [\MaxPosDeviation]
     [\MaxSpeedDeviation]
```

`EGMid`

Datový typ: `egmident`

identita EGM.

`[\Tool]`

Datový typ: `tooldata`

Nástroj použitý pro pohyby prováděné s instrukcí `EGMRunPose`.

Argument `[\Tool]` je volitelný. Výchozí hodnota, když je argument vypuštěn, je `tool0`.

`[\Wobj]`

Datový typ: `wobjdata`

Pracovní objekt použitý pro pohyby prováděné s instrukcí `EGMRunPose`.

Argument `[\Wobj]` je volitelný. Výchozí hodnota, když je argument vypuštěn, je `wobj0`.

`[\TLoad]`

Total load

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Datový typ: `loaddata`

Zatížení použité pro pohyby prováděné s instrukcí `EGMRunPose`.

Argument `[ \TLoad ]` **je volitelný. Výchozí hodnota, když je argument vypuštěn, je** `load0`.

Argument `\TLoad` **popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument** `\TLoad`**, potom** `loaddata` **v aktuálním** `tooldata` **není uvažován.**

Když je argument `\TLoad` **nastaven na** `load0`**, potom není argument** `\TLoad` **uvažován a** `loaddata` **v aktuálním** `tooldata` **je použit místo toho.**

Aby bylo možné použít argument `\TLoad`**, je nezbytné nastavit hodnotu systémového parametru** `ModalPayLoadMode` **na 0. Jestliže** `ModalPayLoadMode` **je nastaven na 0, není dále možné používat instrukci** `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` **nastaven na 0, operátor má možnost kopírovat** `loaddata` **z nástroje do existující nebo nové proměnné perzistentu** `loaddata` **při běhu servisní rutiny.**

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` **(Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1,** `loaddata` **ve volitelném argumentu** `\TLoad` **není uvažován a** `loaddata` **v aktuálním** `tooldata` **se použije místo toho.**

> **POZNÁMKA**
>
> **Výchozí funkčností pro řešení užitečné zátěže je použít instrukci** `GripLoad`. **Proto je výchozí hodnota systémového parametru** `ModalPayLoadMode` **1.**

`CorrFrame`

Datový typ: `pose`

Rámec korekce.

`CorrFrType`

Datový typ: `egmframetype`

Typ rámce korekčního rámce.

`SensorFrame`

Datový typ: `pose`

Rámec senzoru.

`SensFrType`

Datový typ: `egmframetype`

Typ rámce rámce senzoru.

`[\x] [\y] [\z]`

Datový typ: `egm_minmax`

Konvergenční kritérium pro x, y a z v milimetrech. Výchozí hodnota je ± 1,0 milimetru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.65 EGMActPose - Připravit pohyb EGM pro poziční cíl
Externally Guided Motion
Pokračování

Data konvergenčního kritéria se používají pro rozhodnutí, jestli robot dosáhl přikázané pozice v určeném směru osy. Jestliže rozdíl mezi přikázanou pozicí a skutečnou pozicí kloubu je v rozmezí `egm_minmax.min` a `egm_minmax.max`, má se zato, že robot dosáhl své přikázané pozice. Když nejsou určena žádná konvergenční kritéria pro směr osy, který byl zvolen v `EGMRunPose`, použije se výchozí hodnota.

Jakmile všechny osy, které byly určeny v `EGMRunPose`, dosáhly svých přikázaných pozic, samotný robot dosáhl své přikázané pozice a vykonávání RAPID pokračuje s další instrukcí RAPID.

`[\rx] [\ry] [\rz]`

Datový typ: `egm_minmax`

Konvergenční kritérium pro rotaci x, y a z ve stupních. Výchozí hodnota je ± 0,5 stupně.

Data konvergenčního kritéria se používají pro rozhodnutí, jestli robot dosáhl přikázané orientace podél určené osy. Jestliže rozdíl mezi přikázanou orientací a skutečnou orientací je v rozmezí `egm_minmax.min` a `egm_minmax.max`, má se zato, že robot dosáhl své přikázané orientace. Když nejsou určena žádná konvergenční kritéria pro orientaci osy, která byla zvolena v `EGMRunPose`, použije se výchozí hodnota.

Jakmile všechny orientace os, které byly určeny v `EGMRunPose`, dosáhly své přikázané orientace, samotný robot dosáhl své přikázané pozice a vykonávání RAPID pokračuje s další instrukcí RAPID.

`[\LpFilter]`

Datový typ: `num`

Šířka dolní propusti, v Hertzích (Hz), používá se pro filtrování šumu senzoru.

Výchozí hodnota je převzata z konfigurace instrukce `EGMSetupXX`.

`[\SampleRate]`

Datový typ: `num`

Vzorkovací kmitočet načítání vstupních dat v násobcích 4 milisekund. Platné hodnoty jsou 4, 8, 12, 16, atd.

Výchozí hodnota je 4 milisekundy.

`[\MaxPosDeviation]`

Datový typ: `num`

Max. odchylka spoje od naprogramované pozice ve stupních, tj. jemný bod, na kterém začal pohyb EGM. Stejná hodnota se používá pro všechny spoje.

Výchozí hodnota je 1000 stupňů.

`[\MaxSpeedDeviation]`

Datový typ: `num`

Max. přijatelná změna rychlosti spoje ve stupních/sekundách, tzn. že toto je faktor, pomocí kterého se upraví zrychlení/zpomalení.

Výchozí hodnota je 1,0 stupňů/sekund.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

- Jestliže je několikrát proveden `EGMActPose` se stejným `EGMid`, poslední aktivační data se použijí pro instrukce `EGMRunPose`, které následují, dokud běží nový `EGMActPose`.
- `EGMActPose` může být použit pouze v pohybových úlohách RAPID.

Syntaxe

```
EGMActPose
  [EGMid ':='] <variable (VAR) of egmident>
  ['\'Tool ':=' <persistent (PERS) of tooldata>]
  ['\'Wobj ':=' <persistent (PERS) of wobjdata>]
  ['\'TLoad ':=' <persistent (PERS) of loaddata>] ','
  [CorrFrame ':='] < expression (IN) of pose> ','
  [CorrFrType ':='] < expression (IN) of egmframetype> ','
  [SensorFrame ':='] < expression (IN) of pose> ','
  [SensorFrType ':='] < expression (IN) of egmframetype>
  ['\'x ':=' <expression (IN) of egm_minmax>]
  ['\'y ':=' <expression (IN) of egm_minmax>]
  ['\'z ':=' <expression (IN) of egm_minmax>]
  ['\'rx ':=' <expression (IN) of egm_minmax>]
  ['\'ry ':=' <expression (IN) of egm_minmax>]
  ['\'rz ':=' <expression (IN) of egm_minmax>]
  ['\'LpFilter ':=' <expression (IN) of num>]
  ['\'SampleRate ':=' <expression (IN) of num>]
  ['\'MaxPosDeviation ':=' <expression (IN) of num>]
  ['\'MaxSpeedDeviation ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*
Datový typ egm_minmax	*egm_minmax - Konvergenční kritérium pro EGM na str 1490*
Instrukce EGMRunPose	*EGMRunPose - Provést pohyb EGM s pozičním cílem na str 178*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.66 EGMGetId - Dostává identitu EGM

Použití

`EGMGetId` se používá pro rezervaci identity EGM (`EGMid`). Tato identita se potom použije ve všech dalších instrukcích a funkcích EGM RAPID pro identifikaci určitého procesu EGM připojeného k pohybové úloze RAPID, od které je použit.

`egmident` je identifikován podle svého jména, to znamená, druhé nebo třetí volání `EGMGetId` se stejným `egmident` nebude ani rezervovat nový proces EGM ani měnit jeho obsah.

Pro vydání `egmident` k použití jinými procesy EGM se musí použít RAPID instrukce `EGMReset`.

Je možné použít maximálně 4 různé EGM identity současně.

Základní příklady

```
VAR egmident egmID1;
EGMGetId egmID1;
```

Argumenty

```
EGMGetId EGMid
```

`EGMid`

Datový typ: `egmident`

identita EGM.

Omezení

- `EGMGetId` může být použit pouze v pohybových úlohách RAPID.

Syntaxe

```
EGMGetId
  [EGMid ':='] <variable (VAR) of egmident> ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*
Instrukce EGMReset	*EGMReset - Resetovat proces EGM na str 175*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.67 EGMMoveC - Kruhový EGM pohyb s korekcí dráhy

Použití

EGMMoveC se používá k posunu středního bodu nástroje (TCP) kruhově na dané místo určení s korekcí dráhy. Během pohybu zůstává orientace normálně nezměněna ve vztahu ke kruhu.

Základní příklady

Následující příklad názorně ukazuje instrukci EGMMoveC.

Příklad 1

```
VAR egmident EGMid1;
PERS tooldata tReg := [TRUE, [[148,0,326],
     [0.8339007,0,0.551914,0]], [1,[0,0,100], [1,0,0,0], 0,0,0]];
PERS tooldata tLaser := [TRUE, [[148,50,326],
     [0.3902618,-0.589657,-0.589656,0.3902630]],
     [1,[-0.92,0,-0.39], [1,0,0,0], 0,0,0]];
EGMGetId EGMid1;
EGMSetupLTAPP ROB_1, EGMid1, "pathCorr", "OptSim", 1\LATR;
EGMActMove EGMid1, tLaser.tframe\SampleRate:=50;
MoveL p6, v10, fine, tReg\WObj:=wobj0;
EGMMoveL EGMid1, p12, v10, z5, tReg\WObj:=wobj0;
EGMMoveL EGMid1, p7, v10, z5, tReg\WObj:=wobj0;
EGMMoveC EGMid1, p13, p14, v10, z5, tReg\WObj:=wobj0;
EGMMoveL EGMid1, p15, v10, fine, tReg\WObj:=wobj0;
MoveL p8, v1000, z10, tReg\WObj:=wobj0;
EGMReset EGMid1;
```

Tento program registruje proces EGM a nastavuje senzor, který používá komunikační protokol LTAPP a je typu *look-ahead* (výhled) jako datový zdroj (senzor). Senzor by měl používat pro sledování číslo definice 1 typu spoje. Rychlost, kterou řadič přistupuje k zařízení a rámec senzoru zařízení jsou také nastaveny.

Robot se posune na bod startu sledovací dráhy s instrukcí MoveL. Instrukce EGMMove provádějí pohyb robotu s korekcemi od senzoru.

Nakonec je robot přesunut k odchozí pozici a je vydána identita EGM.

Argumenty

```
EGMMoveC EGMid, CirPoint, ToPoint, Speed, Zone, Tool, [\Wobj]
     [\TLoad] [\NoCorr]
```

EGMid

Datový typ: egmident

Identita EGM.

CirPoint

Datový typ:robtarget

Kruhový bod robotu. Kruhový bod je pozice na kruhu mezi start bodem a bodem určení. Aby bylo dosaženo největší přesnosti, měl by být umístěn asi na polovině vzdálenosti mezi body startu a určení. Jestliže je umístěn příliš blízko bodu startu nebo blízko koncového bodu, robot může vydat varování. Střední bod je definován

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci). Pozice externích os se nepoužívají.

`ToPoint`

Datový typ:`robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`Zone`

Datový typ:`zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného pozice.

`[\WObj]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadný systém objektu), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnicovému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být stanoven kvůli kruhu ve vztahu k pracovnímu objektu, který bude proveden.

`[\TLoad]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Je možné nechat program běžet zkušebně bez užitečné zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

`[\NoCorr]`

Datový typ: `switch`

Korekce dráhy je vypnuta.

Vykonávání programu

`EGMMoveC` vede robot podél naprogramované kruhové dráhy s přenesenými korekcemi od senzoru. Během tohoto pohybu vyžaduje instrukce korekční data od senzoru při rychlosti nastavené s `EGMActMove`. Jestliže je přítomen volitelný argument `\NoCorr`, do naprogramované dráhy se nepřidává žádná korekce.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_UDPUC_COMM`	Chyba vznikla v komunikaci se zařízením UdpUc.

Omezení

- `EGMMoveC` může být použit pouze v pohybových úlohách RAPID.

Syntaxe

```
EGMMoveC
  [GMid ':='] <variable (VAR) of egmident> ','
  [CirPoint ':='] < expression (IN) of robtarget> ','
  [ToPoint ':='] < expression (IN) of robtarget> ','
  [Speed ':='] < expression (IN) of speeddata> ','
  [Zone ':='] < expression (IN) of zonedata> ','
  [Tool ':='] < persistent (PERS) of tooldata>
  ['\'WObj ':=' < persistent (PERS) of wobjdata>]
  ['\'TLoad ':=' < persistent (PERS) of loaddata>]
  ['\'NoCorr] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.68 EGMMoveL - Lineární EGM pohyb s korekcí dráhy

Použití

EGMMoveL se používá k posunutí středního bodu nástroje (TCP) lineárně k danému bodu určení s korekcí dráhy. Jestliže TCP má zůstat stacionární, potom se tato instrukce může použít také pro reorientaci nástroje.

Základní příklady

Následující příklad názorně ukazuje instrukci EGMMoveL.

Příklad 1

```
VAR egmident EGMid1;
PERS tooldata tReg := [TRUE, [[148,0,326],
     [0.8339007,0,0.551914,0]], [1,[0,0,100], [1,0,0,0], 0,0,0]];
PERS tooldata tLaser := [TRUE, [[148,50,326],
     [0.3902618,-0.589657,-0.589656,0.3902630]],
     [1,[-0.92,0,-0.39], [1,0,0,0], 0,0,0]];
EGMGetId EGMid1;
EGMSetupLTAPP ROB_1, EGMid1, "pathCorr", "OptSim", 1\LATR;
EGMActMove EGMid1, tLaser.tframe\SampleRate:=50;
MoveL p6, v10, fine, tReg\WObj:=wobj0;
EGMMoveL EGMid1, p12, v10, z5, tReg\WObj:=wobj0;
EGMMoveL EGMid1, p7, v10, z5, tReg\WObj:=wobj0;
EGMMoveC EGMid1, p13, p14, v10, z5, tReg\WObj:=wobj0;
EGMMoveL EGMid1, p15, v10, fine, tReg\WObj:=wobj0;
MoveL p8, v1000, z10, tReg\WObj:=wobj0;
EGMReset EGMid1;
```

Tento program registruje proces EGM a nastavuje senzor, který používá komunikační protokol LTAPP a je typu *look-ahead* (výhled) jako datový zdroj (senzor). Senzor by měl používat pro sledování číslo definice 1 typu spoje. Rychlost, kterou řadič přistupuje k zařízení a rámec senzoru zařízení jsou také nastaveny.

Robot se posune na bod startu sledovací dráhy s instrukcí MoveL. Instrukce EGMMove provádějí pohyb robotu s korekcemi od senzoru.

Nakonec je robot přesunut k odchozí pozici a je vydána identita EGM.

Argumenty

```
EGMMoveL EGMid, ToPoint, Speed, Zone, Tool, [\Wobj] [\TLoad]
     [\NoCorr]
```

EGMid

Datový typ: egmident

Identita EGM.

ToPoint

Datový typ:robtarget

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`Zone`

Datový typ:`zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného pozice.

`[\WObj]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadný systém objektu), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnicovému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být stanoven kvůli kruhu ve vztahu k pracovnímu objektu, který bude proveden.

`[\TLoad]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` **popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument** `\TLoad`**, potom** `loaddata` **v aktuálním** `tooldata` **není uvažován.**

Když je argument `\TLoad` **nastaven na** `load0`**, potom není argument** `\TLoad` **uvažován a** `loaddata` **v aktuálním** `tooldata` **je použit místo toho.**

Aby bylo možné použít argument `\TLoad`**, je nezbytné nastavit hodnotu systémového parametru** `ModalPayLoadMode` **na 0. Jestliže** `ModalPayLoadMode` **je nastaven na 0, není dále možné používat instrukci** `GripLoad`**.**

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` **nastaven na 0, operátor má možnost kopírovat** `loaddata` **z nástroje do existující nebo nové proměnné perzistentu** `loaddata` **při běhu servisní rutiny.**

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` **(Simulated Mode).**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.68 EGMMoveL - Lineární EGM pohyb s korekcí dráhy

Externally Guided Motion
Pokračování

Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

> **POZNÁMKA**
>
> Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

`[\NoCorr]`

Datový typ: `switch`

Korekce dráhy je vypnuta.

Vykonávání programu

`EGMMoveL` vede robot podél naprogramované lineární dráhy s přenesenými korekcemi od senzoru. Během tohoto pohybu vyžaduje instrukce korekční data od senzoru při rychlosti nastavené s `EGMActMove`. Jestliže je přítomen volitelný argument `\NoCorr`, do naprogramované dráhy se nepřidává žádná korekce.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_UDPUC_COMM`	Chyba vznikla v komunikaci se zařízením UdpUc.

Omezení

- `EGMMoveL` může být použit pouze v pohybových úlohách RAPID.

Syntaxe

```
EGMMoveL
  [EGMid ':='] <variable (VAR) of egmident> ','
  [ToPoint ':='] < expression (IN) of robtarget> ','
  [Speed ':='] < expression (IN) of speeddata> ','
  [Zone ':='] < expression (IN) of zonedata> ','
  [Tool ':='] < persistent (PERS) of tooldata>
  ['\'WObj ':=' < persistent (PERS) of wobjdata>]
  ['\'TLoad ':=' < persistent (PERS) of loaddata>]
  ['\'NoCorr] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.69 EGMReset - Resetovat proces EGM

Použití

`EGMReset` **resetuje specifický proces EGM** (`EGMid`)**, to znamená, že rezervace je zrušena.**

Základní příklady

```
VAR egmident egmID1;
PERS pose pose1:=[[0,0,0], [1,0,0,0]];
CONST egm_minmax egm_minmax_lin:=[-0.1,0.1];
CONST egm_minmax egm_minmax_rot:=[-0.1,0.2];
CONST pose posecor:=[[1200,400,900], [0,0,1,0]];
CONST pose posesens:=[[12.3313,-0.108707,416.142],
     [0.903899,-0.00320735,0.427666,0.00765917]];

EGMGetId egmID1;
EGMSetupAI ROB_1, egmID1, "default" \Pose \aiR1x:=ai_01
     \aiR2y:=ai_02 \aiR3z:=ai_03 \aiR4rx:=ai_04 \aiR5ry:=ai_05
     \aiR6rz:=ai_06;
EGMActPose egmID1 \Tool:=tool0 \WObj:=wobj0, posecor,
     EGM_FRAME_WOBJ, posesens, EGM_FRAME_TOOL \x:=egm_minmax_lin
     \y:=egm_minmax_lin \z:=egm_minmax_lin \rx:=egm_minmax_rot
     \ry:=egm_minmax_rot \rz:=egm_minmax_rot \LpFilter:=20;
EGMRunPose egmID1, EGM_STOP_HOLD \x \y \z \rx \ry \rz
     \RampInTime:=0.05;
EGMReset egmID1;
```

Argumenty

```
EGMReset EGMid
```

`EGMid`

Datový typ: `egmident`

identita EGM.

Syntaxe

```
EGMReset
  [EGMid ':='] <variable (VAR) of egmident>';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.70 EGMRunJoint - Provést pohyb EGM s cílem spoje

Použití

`EGMRunJoint` provádí pohyb vedený senzorem k cíli spoje od jemného bodu v konkrétním procesu EGM (`EGMid`) a definuje, které spoje budou posunuty.

Základní příklady

```
VAR egmident egmID1;
PERS pose pose1:=[[0,0,0],[1,0,0,0]];
CONST egm_minmax egm_minmax1:=[-1,1];

EGMGetId egmID1;
EGMSetupAI ROB_1, egmID1, "default" \Joint \aiR1x:=ai_01
     \aiR2y:=ai_02 \aiR3z:=ai_03 \aiR4rx:=ai_04 \aiR5ry:=ai_05
     \aiR6rz:=ai_06;
EGMActJoint egmID1, \J1:=egm_minmax1 \J3:=egm_minmax1
     \J4:=egm_minmax1;
EGMRunJoint egmID1, EGM_STOP_HOLD \J1 \J3 \RampInTime:=0.05;
```

Argumenty

```
EGMRunJoint EGMid, Mode [\J1] [\J2] [\J3] [\J4] [\J5] [\J6]
     [\CondTime] [\RampInTime] [\RampOutTime] [\PosCorrGain]
```

`EGMid`

Datový typ: `egmident`

identita EGM.

`Mode`

Datový typ: `egmstopmode`

Definuje způsob zakončení pohybu (`EGM_STOP_HOLD, EGM_STOP_RAMP_DOWN`)

`[\J1] [\J2] [\J3] [\J4] [\J5] [\J6]`

Datový typ: `switch`

Posunout spoj 1 až 6

`[\CondTime]`

Datový typ: `num`

Čas v sekundách, kdy konvergenční kritérium definované v `EGMActJoint` musí být splněno předtím, než je cílový bod považován za dosažený a `EGMRunJoint` vydává provedení RAPID k pokračování k další instrukci.

Výchozí hodnota je 1 sek.

`[\RampInTime]`

Datový typ: `num`

Definuje v sekundách, jak rychle je pohyb spuštěn.

`[\RampOutTime]`

Datový typ: `num`

Definuje v sekundách, jak rychle bude provedeno EGM rampa dolů.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Tento parametr nemá žádný význam, jestliže parametr `Mode` je nastaven na `EGM_STOP_HOLD`.

`[\PosCorrGain]`

Datový typ: `num`

Přírůstek korekce pozice. Hodnota mezi 0 a 1, výchozí je 1.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_UDPUC_COMM`	Chyba vznikla v komunikaci se zařízením UdpUc.

Omezení

- Před prvním použitím `EGMRunJoint` musí být robot posunut po spuštění ovladače. Buď krokově nebo vykonáním instrukcí Move od RAPIDu.
- Bod startu pro pohyb `EGMRunJoint` musí být jemným bodem.
- `EGMRunJoint` může být použit pouze v pohybových úlohách RAPID.
- Jestliže byla provedena instrukce `EGMActPose` namísto `EGMActJoint`, objeví se následující chyba: *41826 EGM mode mismatch*.
- Jestliže není upřesněn žádný z přepínačů `\J1` až `\J6`, neprovede se žádný pohyb a vykonávání RAPID pokračuje k další instrukci RAPID.

Syntaxe

```
EGMRunJoint
  [EGMid ':='] <variable (VAR) of egmident> ','
  [Mode ':='] < expression (IN) of egmstopmode>
  ['\'J1]
  ['\'J2]
  ['\'J3]
  ['\'J4]
  ['\'J5]
  ['\'J6]
  ['\'CondTime ':=' <expression (IN) of num>]
  ['\'RampInTime ':=' <expression (IN) of num>]
  ['\'RampOutTime ':=' <expression (IN) of num>]
  ['\'PosCorrGain ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*
Datový typ egmstopmode	*egmstopmode - Definuje stop režimy pro EGM na str 1492*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.71 EGMRunPose - Provést pohyb EGM s pozičním cílem

Použití

`EGMRunPose` provádí pohyb vedený senzorem k pozičnímu cíli od jemného bodu v konkrétním procesu EGM (`EGMid`) a definuje, které směry a orientace mohou být změněny.

Základní příklady

```
VAR egmident egmID1;
PERS pose pose1:=[[0,0,0],[1,0,0,0]];
CONST egm_minmax egm_minmax_lin:=[-0.1,0.1];
CONST egm_minmax egm_minmax_rot:=[-0.1,0.2];
CONST pose posecor:=[[1200,400,900],[0,0,1,0]];
CONST pose posesens:=[[12.3313,-0.108707,416.142],
     [0.903899,-0.00320735,0.427666,0.00765917]];

EGMGetId egmID1;
EGMSetupAI ROB_1, egmID1, "default" \Pose \aiR1x:=ai_01
     \aiR2y:=ai_02 \aiR3z:=ai_03 \aiR4rx:=ai_04 \aiR5ry:=ai_05
     \aiR6rz:=ai_06;
EGMActPose egmID1 \Tool:=tool0 \WObj:=wobj0, posecor,
     EGM_FRAME_WOBJ, posesens, EGM_FRAME_TOOL \x:=egm_minmax_lin
     \y:=egm_minmax_lin \z:=egm_minmax_lin \rx:=egm_minmax_rot
     \ry:=egm_minmax_rot \rz:=egm_minmax_rot \LpFilter:=20;
EGMRunPose egmID1, EGM_STOP_HOLD \x \y \z \rx \ry \rz
     \RampInTime:=0.05;
```

Argumenty

```
EGMRunPose EGMid, Mode [\x] [\y] [\z] [\rx] [\ry] [\rz] [\CondTime]
     [\RampInTime] [\RampOutTime] [\Offset] [\PosCorrGain]
```

`EGMid`

Datový typ: `egmident`

identita EGM.

`Mode`

Datový typ: `egmstopmode`

Definuje způsob zakončení pohybu (`EGM_STOP_HOLD, EGM_STOP_RAMP_DOWN`)

`[\x] [\y] [\z]`

Datový typ: `switch`

Pohyb ve směru x, y a z.

`[\rx] [\ry] [\rz]`

Datový typ: `switch`

Reorientace kolem os x, y a z.

`[\CondTime]`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Čas v sekundách, kdy konvergenční kritérium definované v EGMActPose musí být splněno předtím, než je cílový bod považován za dosažený a EGMRunPose vydává provedení RAPID k pokračování k další instrukci.

Výchozí hodnota je 1 sek.

[\RampInTime]

Datový typ: num

Definuje v sekundách, jak rychle je pohyb spuštěn.

[\RampOutTime]

Datový typ: num

Definuje v sekundách, jak rychle bude provedeno EGM rampa dolů.

Tento parametr nemá žádný význam, jestliže parametr Mode je nastaven na EGM_STOP_HOLD.

[\Offset]

Datový typ: pose

Možnost definovat statický ofset na vrcholu hodnoty dané senzorem.

[\PosCorrGain]

Datový typ: num

Přírůstek korekce pozice. Hodnota mezi 0 a 1, výchozí je 1.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_UDPUC_COMM	Chyba vznikla v komunikaci se zařízením UdpUc.

Omezení

- Před prvním použitím EGMRunPose musí být robot posunut po spuštění ovladače. Buď krokově nebo vykonáním instrukcí Move od RAPIDu.
- Bod startu pro pohyb EGMRunPose musí být jemným bodem.
- EGMRunPose může být použit pouze v pohybových úlohách RAPID.
- Jestliže byla provedena instrukce EGMActJoint namísto EGMRunPose, objeví se následující chyba: *41826 EGM mode mismatch*.
- Jestliže není upřesněn žádný z přepínačů \x až \rz, neprovede se žádný pohyb a vykonávání RAPID pokračuje k další instrukci RAPID.

Syntaxe

```
EGMRunPose
  [EGMid ':='] <variable (VAR) of egmident>','
  [Mode ':='] < expression (IN) of egmstopmode>
  ['\'x]
  ['\'y]
  ['\'z]
  ['\'rx]
  ['\'ry]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.71 EGMRunPose - Provést pohyb EGM s pozičním cílem

Externally Guided Motion

Pokračování

```
['\'rz]
['\'CondTime ':=' <expression (IN) of num>]
['\'RampInTime ':=' <expression (IN) of num>]
['\'RampOutTime ':=' <expression (IN) of num>]
['\'Offset ':=' <expression (IN) of pose>]
['\'PosCorrGain ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*
Datový typ egmstopmode	*egmstopmode - Definuje stop režimy pro EGM na str 1492*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.72 EGMSetupAI - Nastavit analogové vstupní signály pro EGM

Použití

`EGMSetupAI` se používá pro nastavování analogových vstupních signálů pro konkrétní proces EGM (`EGMid`), jako zdroj pro poziční cílové hodnoty, na které bude naveden robot a až 6 pomocných os.

Základní příklady

```
VAR egmident egmID1;

EGMGetId egmID1;
EGMSetupAI ROB_1, egmID1, "default" \Pose \aiR1x:=ai_01
     \aiR2y:=ai_02 \aiR3z:=ai_03 \aiR4rx:=ai_04 \aiR5ry:=ai_05
     \aiR6rz:=ai_06;
```

Argumenty

```
EGMSetupAI MecUnit, EGMid, ExtConfigName [\Joint] | [\Pose] |
     [\PathCorr] [\APTR] | [\LATR] [\aiR1x] [\aiR2y] [\aiR3z]
     [\aiR4rx] [\aiR5ry] [\aiR6rz] [\aiE1] [\aiE2] [\aiE3] [\aiE4]
     [\aiE5] [\aiE6]
```

`MecUnit`

Datový typ: `mecunit`

Název mechanické jednotky.

`EGMid`

Datový typ: `egmident`

identita EGM.

`ExtConfigName`

Datový typ: `string`

Jméno dat externího rozhraní pohybu, jak je definováno v systémových parametrech.

Více informací najdete v *Technická referenční příručka - Systémové parametry*, napište *External Motion Interface Data*, téma *Motion*.

`[\Joint]`

Datový typ: `switch`

Volí pohyb spoje pro poziční navádění.

Alespoň jeden z přepínačů `\Joint`, `\Pose`, or `\PathCorr` musí být přítomen.

`[\Pose]`

Datový typ: `switch`

Volí poziční pohyb pro poziční navádění.

Alespoň jeden z přepínačů `\Joint`, `\Pose`, or `\PathCorr` musí být přítomen.

`[\PathCorr]`

Datový typ: `switch`

Volí korekci dráhy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.72 EGMSetupAI - Nastavit analogové vstupní signály pro EGM
Externally Guided Motion
Pokračování

Alespoň jeden z přepínačů \Joint, \Pose, or \PathCorr musí být přítomen.

[\APTR]

Datový typ: switch

Nastavit senzor typu bodového sledovače pro korekci dráhy. Například WeldGuide nebo AWC.

Buď \APTR nebo \LATR musí být přítomen.

[\LATR]

Datový typ: switch

Nastavit senzor typu dopředného sledovače pro korekci dráhy. Například Laser Tracker.

Buď \APTR nebo \LATR musí být přítomen.

[\aiR1x] [\aiR2y] [\aiR3z]

Datový typ: signalai

Určuje signál, který poskytuje hodnotu x, y a z pro poziční pohyb.

Určuje signál, který dodává úhel spojů robotu 1 až 3 ve stupních pro pohyb spoje.

[\aiR4rx] [\aiR5ry] [\aiR6rz]

Datový typ: signalai

Určuje signál, který dodává hodnotu rotace x, y a z robotu ve stupních pro poziční pohyb.

Určuje signál, který dodává úhel spojů robotu 4 až 6 ve stupních pro pohyb spoje.

[\aiE1] [\aiE2] [\aiE3] [\aiE4] [\aiE5] [\aiE6]

Datový typ: signalai

Určuje signál, který dodává pozici spojů 1 až 6 pomocné osy.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_NO_ALIASIO_DEF	Proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.
ERR_NORUNUNIT	Není žádný kontakt s jednotkou I/O.
ERR_SIG_NOT_VALID	Není přístup k I/O signálu (platí pouze pro aplikační sběrnici ICI).

Omezení

- EGMSetupAI může být použit pouze v pohybových úlohách RAPID.
- Mechanická jednotka musí být robot.
- Alespoň jeden signál je třeba určit, jinak bude odeslána chyba a vykonávání RAPID se zastaví.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
EGMSetupAI
  [MecUnit ':='] <variable (VAR) of mecunit> ’,'
  [EGMid ':='] <variable (VAR) of egmident> ','
  [ExtConfigName ':='] <expression (IN) of string>
  [['\'Joint] | ['\'Pose] | ['\'PathCorr ]]
  [['\'APTR] | ['\'LAT]] ','
  ['\'aiR1x ':=' <variable (VAR) of signalai>]
  ['\'aiR2y ':=' <variable (VAR) of signalai>]
  ['\'aiR3z ':=' <variable (VAR) of signalai>]
  ['\'aiR4rx ':=' <variable (VAR) of signalai>]
  ['\'aiR5ry ':=' <variable (VAR) of signalai>]
  ['\'aiR6rz ':=' <variable (VAR) of signalai>]
  ['\'aiE1 ':=' <variable (VAR) of signalai>]
  ['\'aiE2 ':=' <variable (VAR) of signalai>]
  ['\'aiE3 ':=' <variable (VAR) of signalai>]
  ['\'aiE4 ':=' <variable (VAR) of signalai>]
  ['\'aiE5 ':=' <variable (VAR) of signalai>]
  ['\'aiE6 ':=' <variable (VAR) of signalai>] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.73 EGMSetupAO - Nastavit analogové výstupní signály pro EGM

Použití

`EGMSetupAO` se používá pro nastavování AO signálů pro konkrétní proces EGM (`EGMid`), jako zdroj pro poziční cílové hodnoty, na které bude naveden robot a až 6 pomocných os.

Základní příklady

```
VAR egmident egmID1;

EGMGetId egmID1;
EGMSetupAO ROB_1, egmID1, "default" \Pose \aoR1x:=ao_01
     \aoR2y:=ao_02 \aoR3z:=ao_03 \aoR4rx:=ao_04 \aoR5ry:=ao_05
     \aoR6rz:=ao_06;
```

Argumenty

```
EGMSetupAO MecUnit, EGMid, ExtConfigName [\Joint] | [\Pose] |
     [\PathCorr] [\APTR] | [\LATR] [\aoR1x] [\aoR2y] [\aoR3Z]
     [\aoR4rx] [\aoR5ry] [\aoR6rz] [\aoE1] [\aoE2] [\aoE3] [\aoE4]
     [\aoE5] [\aoE6]
```

`MecUnit`

Datový typ: `mecunit`

Název mechanické jednotky.

`EGMid`

Datový typ: `egmident`

identita EGM.

`ExtConfigName`

Datový typ: `string`

Jméno dat externího rozhraní pohybu, jak je definováno v systémových parametrech.

Více informací najdete v *Technická referenční příručka - Systémové parametry*, napište *External Motion Interface Data*, téma *Motion*.

`[\Joint]`

Datový typ: `switch`

Volí pohyb spoje.

Alespoň jeden z přepínačů `\Joint` or `\Pose` musí být přítomen.

`[\Pose]`

Datový typ: `switch`

Volí poziční pohyb.

Alespoň jeden z přepínačů `\Joint` or `\Pose` musí být přítomen.

`[\PathCorr]`

Datový typ: `switch`

Volí korekci dráhy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Alespoň jeden z přepínačů `\Joint`, `\Pose`, **or** `\PathCorr` **musí být přítomen.**

`[\APTR]`

Datový typ: `switch`

Nastavit senzor typu bodového sledovače pro korekci dráhy. Například WeldGuide nebo AWC.

Buď `\APTR` **nebo** `\LATR` **musí být přítomen.**

`[\LATR]`

Datový typ: `switch`

Nastavit senzor typu dopředného sledovače pro korekci dráhy. Například Laser Tracker.

Buď `\APTR` **nebo** `\LATR` **musí být přítomen.**

`[\aoR1x] [\aoR2y] [\aoR3z]`

Datový typ: `signalao`

Určuje signál, který poskytuje hodnotu x, y a z pro poziční pohyb.

Určuje signál, který dodává úhel spojů robotu 1 až 3 ve stupních pro pohyb spoje.

`[\aoR4rx] [\aoR5ry] [\aoR6rz]`

Datový typ: `signalao`

Určuje signál, který dodává hodnotu rotace x, y a z robotu ve stupních pro poziční pohyb.

Určuje signál, který dodává úhel spojů robotu 4 až 6 ve stupních pro pohyb spoje.

`[\aoE1] [\aoE2] [\aoE3] [\aoE4] [\aoE5] [\aoE6]`

Datový typ: `signalao`

Určuje signál, který dodává pozici spojů 1 až 6 pomocné osy.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` **bude nastavena na:**

Název	Příčina chyby
`ERR_NO_ALIASIO_DEF`	Proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.
`ERR_NORUNUNIT`	Není žádný kontakt s jednotkou I/O.
`ERR_SIG_NOT_VALID`	Není přístup k I/O signálu (platí pouze pro aplikační sběrnici ICI).

Omezení

- `EGMSetupAO` **může být použit pouze v pohybových úlohách RAPID.**
- **Mechanická jednotka musí být robot.**
- **Alespoň jeden signál je třeba určit, jinak bude odeslána chyba a vykonávání RAPID se zastaví.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.73 EGMSetupAO - Nastavit analogové výstupní signály pro EGM

Externally Guided Motion
Pokračování

Syntaxe

```
EGMSetupAO
  [MecUnit ':='] <variable (VAR) of mecunit> ','
  [EGMid ':='] <variable (VAR) of egmident> ','
  [ExtConfigName ':='] <expression (IN) of string>
  [['\'Joint] | ['\'Pose] | ['\'PathCorr]]
  [['\'APTR] | ['\'LATR]]
  ['\'aoR1x ':=' <variable (VAR) of signalao>]
  ['\'aoR2y ':='] <variable (VAR) of signalao>]
  ['\'aoR3z ':='] <variable (VAR) of signalao>]
  ['\'aoR4rx ':='] <variable (VAR) of signalao>]
  ['\'aoR5ry ':='] <variable (VAR) of signalao>]
  ['\'aoR6rz ':='] <variable (VAR) of signalao>]
  ['\'aoE1 ':='] <variable (VAR) of signalao>]
  ['\'aoE2 ':='] <variable (VAR) of signalao>]
  ['\'aoE3 ':='] <variable (VAR) of signalao>]
  ['\'aoE4 ':='] <variable (VAR) of signalao>]
  ['\'aoE5 ':='] <variable (VAR) of signalao>]
  ['\'aoE6 ':='] <variable (VAR) of signalao>] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.74 EGMSetupGI - Nastavit skupinové vstupní signály pro EGM

Použití

`EGMSetupGI` se používá pro nastavování skupinových vstupních signálů pro konkrétní proces EGM (`EGMid`), jako zdroj pro poziční cílové hodnoty, na které bude naveden robot a až 6 pomocných os.

Základní příklady

```
VAR egmident egmID1;

EGMGetId egmID1;
EGMSetupGI ROB_1, egmID1, "default" \Pose \giR1x:=gi_01
     \giR2y:=gi_02 \giR3z:=gi_03 \giR4rx:=gi_04 \giR5ry:=gi_05
     \giR6rz:=gi_06;
```

Argumenty

```
EGMSetupGI MecUnit, EGMid, ExtConfigName [\Joint] | [\Pose] |
     [\PathCorr] [\APTR] | [\LATR] [\giR1x] [\giR2y] [\giR3Z]
     [\giR4rx] [\giR5ry] [\giR6rz] [\giE1] [\giE2] [\giE3] [\giE4]
     [\giE5] [\giE6]
```

`MecUnit`

Datový typ: `mecunit`

Název mechanické jednotky.

`EGMid`

Datový typ: `egmident`

identita EGM.

`ExtConfigName`

Datový typ: `string`

Jméno dat externího rozhraní pohybu, jak je definováno v systémových parametrech.

Více informací najdete v *Technická referenční příručka - Systémové parametry*, napište *External Motion Interface Data*, téma *Motion*.

`[\Joint]`

Datový typ: `switch`

Volí pohyb spoje.

Alespoň jeden z přepínačů `\Joint` or `\Pose` musí být přítomen.

`[\Pose]`

Datový typ: `switch`

Volí poziční pohyb.

Alespoň jeden z přepínačů `\Joint` or `\Pose` musí být přítomen.

`[\PathCorr]`

Datový typ: `switch`

Volí korekci dráhy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.74 EGMSetupGI - Nastavit skupinové vstupní signály pro EGM

Externally Guided Motion

Pokračování

Alespoň jeden z přepínačů `\Joint`, `\Pose`, or `\PathCorr` musí být přítomen.

`[\APTR]`

Datový typ: `switch`

Nastavit senzor typu bodového sledovače pro korekci dráhy. Například WeldGuide nebo AWC.

Buď `\APTR` nebo `\LATR` musí být přítomen.

`[\LATR]`

Datový typ: `switch`

Nastavit senzor typu dopředného sledovače pro korekci dráhy. Například Laser Tracker.

Buď `\APTR` nebo `\LATR` musí být přítomen.

`[\giR1x] [\giR2y] [\giR3z]`

Datový typ: `signalgi`

Určuje signál, který poskytuje hodnotu x, y a z pro poziční pohyb.

Určuje signál, který dodává úhel spojů robotu 1 až 3 ve stupních pro pohyb spoje.

`[\giR4rx] [\giR5ry] [\giR6rz]`

Datový typ: `signalgi`

Určuje signál, který dodává hodnotu rotace x, y a z robotu ve stupních pro poziční pohyb.

Určuje signál, který dodává úhel spojů robotu 4 až 6 ve stupních pro pohyb spoje.

`[\giE1] [\giE2] [\giE3] [\giE4] [\giE5] [\giE6]`

Datový typ: `signalgi`

Určuje signál, který dodává pozici spojů 1 až 6 pomocné osy.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_NO_ALIASIO_DEF`	Proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.
`ERR_NORUNUNIT`	Není žádný kontakt s jednotkou I/O.
`ERR_SIG_NOT_VALID`	Není přístup k I/O signálu (platí pouze pro aplikační sběrnici ICI).

Omezení

- `EGMSetupGI` **může být použit pouze v pohybových úlohách RAPID.**
- **Mechanická jednotka musí být robot.**
- **Skupinové signály mohou řešit pouze kladné hodnoty. Proto je jejich využití v EGM omezeno.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

- Alespoň jeden signál je třeba určit, jinak bude odeslána chyba a vykonávání RAPID se zastaví.

Syntaxe

```
EGMSetupGI
  [MecUnit ':='] <variable (VAR) of mecunit> ','
  [EGMid ':='] <variable (VAR) of egmident> ','
  [ExtConfigName ':='] <expression (IN) of string>
  [['\'Joint] | ['\'Pose] | ['\'PathCorr]]
  [['\'APTR] | ['\'LATR]]
  ['\'giR1x ':=' <variable (VAR) of signalgi>]
  ['\'giR2y ':=' <variable (VAR) of signalgi>]
  ['\'giR3z ':=' <variable (VAR) of signalgi>]
  ['\'giR4rx ':=' <variable (VAR) of signalgi>]
  ['\'giR5ry ':=' <variable (VAR) of signalgi>]
  ['\'giR6rz ':=' <variable (VAR) of signalgi>]
  ['\'giE1 ':=' <variable (VAR) of signalgi>]
  ['\'giE2 ':=' <variable (VAR) of signalgi>]
  ['\'giE3 ':=' <variable (VAR) of signalgi>]
  ['\'giE4 ':=' <variable (VAR) of signalgi>]
  ['\'giE5 ':=' <variable (VAR) of signalgi>]
  ['\'giE6 ':=' <variable (VAR) of signalgi>] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.75 EGMSetupLTAPP - Nastavit LTAPP protokol pro EGM

Použití

`EGMSetupLTAPP` se používá pro nastavování protokolu *LTAPP* pro specifický proces EGM (`EGMid`) jako zdroj korekcí dráhy.

Základní příklady

Následující příklad názorně ukazuje instrukci `EGMSetupLTAPP`.

Příklad 1

```
VAR egmident EGMid1;
EGMGetId EGMid1;
EGMSetupLTAPP ROB_1, EGMid1, "pathCorr", "OptSim", 1\LATR;
```

Tento program registruje proces EGM a nastavuje `OptSim` senzoru, který používá komunikační protokol LTAPP a je typu *look-ahead* (dopředný) jako datový zdroj (senzor). Senzor by měl používat pro sledování číslo definice 1 typu spoje.

Argumenty

```
EGMActMove MecUnit, EGMid, ExtConfigName, Device, JointType [\APTR]
     | [\LATR]
```

`MecUnit`

Datový typ: `mecunit`

Název mechanické jednotky.

`EGMid`

Datový typ: `egmident`

Identita EGM.

`ExtConfigName`

Datový typ: `string`

Jméno dat externího rozhraní pohybu, jak je definováno v systémových parametrech.

Další informace obsahuje *Technická referenční příručka - Systémové parametry*, téma *Motion*, typ *External Motion Interface Data*.

`Device`

Datový typ: `string`

Jméno zařízení LTAPP.

`JointType`

Datový typ: `num`

Definuje typ spoje, vyjádřeného jako číslo, který by mělo používat zařízení senzoru během korekce dráhy.

`[\APTR]`

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Nastavit senzor typu bodového sledovače pro korekci dráhy. Například WeldGuide nebo AWC.

Buď \APTR nebo \LATR musí být přítomen.

[\LATR]

Datový typ: switch

Nastavit senzor typu dopředného sledovače pro korekci dráhy. Například Laser Tracker.

Buď \APTR nebo \LATR musí být přítomen.

Vykonávání programu

EGMSetupLTAPP připojuje charakteristická data použitého senzoru k identitě EGM. Tato identita EGM může být potom použita v různých instrukcích EGMActMove a EGMMove.

Syntaxe

```
EGMSetupLTAPP
  [MecUnit ':='] <variable (VAR) of mecunit> ','
  [EGMid ':='] <variable (VAR) of egmident> ','
  [ExtConfigName ':='] < expression (IN) of string> ','
  [Device ':='] < expression (IN) of string> ','
  [JointType ':='] < expression (IN) of num>
  [['\'APTR] | ['\'LATR]] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.76 EGMSetupUC - Nastavit protokol UdpUc pro EGM

Použití

`EGMSetupUC` se používá pro nastavování protokolu UdpUc pro konkrétní proces EGM (`EGMid`) jako zdroj pro poziční cílové hodnoty, na které bude naveden robot a až 6 pomocných os.

Základní příklady

```
VAR egmident egmID1;
VAR string egmSensor:="egmSensor:";
EGMGetId egmID1;
EGMSetupUC ROB_1, egmID1, "default", egmSensor\Pose;
```

Argumenty

```
EGMSetupUC MecUnit, EGMid, ExtConfigName, UCDevice [\Joint] |
     [\Pose] | [\PathCorr] [\APTR] | [\LATR] [\CommTimeout]
```

`MecUnit`

Datový typ: `mecunit`

Název mechanické jednotky.

`EGMid`

Datový typ: `egmident`

identita EGM.

`ExtConfigName`

Datový typ: `string`

Jméno dat externího rozhraní pohybu, jak je definováno v systémových parametrech.

Více informací najdete v *Technická referenční příručka - Systémové parametry*, napište *External Motion Interface Data*, téma *Motion*.

`UCDevice`

Datový typ: `string`

Jméno zařízení UdpUc.

`[\Joint]`

Datový typ: `switch`

Volí pohyb spoje pro poziční navádění.

Alespoň jeden z přepínačů `\Joint`, `\Pose`, or `\PathCorr` musí být přítomen.

`[\Pose]`

Datový typ: `switch`

Volí poziční pohyb pro poziční navádění.

Alespoň jeden z přepínačů `\Joint`, `\Pose`, or `\PathCorr` musí být přítomen.

`[\PathCorr]`

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Volí korekci dráhy.

Alespoň jeden z přepínačů `\Joint`, `\Pose`, or `\PathCorr` musí být přítomen.

`[\APTR]`

Datový typ: `switch`

Nastavit senzor typu bodového sledovače pro korekci dráhy. Například WeldGuide nebo AWC.

Buď `\APTR` nebo `\LATR` musí být přítomen.

`[\LATR]`

Datový typ: `switch`

Nastavit senzor typu dopředného sledovače pro korekci dráhy. Například Laser Tracker.

Buď `\APTR` nebo `\LATR` musí být přítomen.

`[\CommTimeout]`

Datový typ: `num`

Stanovený časový úsek pro komunikaci s externím zařízením UdpUC v sekundách.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_UDPUC_COMM`	Chyba vznikla v komunikaci se zařízením UdpUc.

Omezení

- `EGMSetupUC` může být použit pouze v pohybových úlohách RAPID.
- Mechanická jednotka musí být robot.

Syntaxe

```
EGMSetupUC
  [MecUnit ':='] <variable (VAR) of mecunit> ','
  [EGMid ':='] <variable (VAR) of egmident> ','
  [ExtConfigName ':='] <expression (IN) of string> ','
  [UCDevice ':='] <expression (IN) of string>
  [['\'Joint] | ['\'Pose] | ['\'PathCorr]]
  [['\'APTR] | ['\'LATR]]
  ['\'CommTimeout ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.77 EGMStop - Zastavit pohyb EGM

Použití

`EGMStop` zastavuje specifický proces EGM (`EGMid`).

Základní příklady

V pohybové úloze RAPID:

```
VAR egmident egmID1;
PERS pose pose1:=[[0,0,0], [1,0,0,0]];
CONST egm_minmax egm_minmax_lin:=[-0.1,0.1];
CONST egm_minmax egm_minmax_rot:=[-0.1,0.2];
CONST pose posecor:=[[1200,400,900], [0,0,1,0]];
CONST pose posesens:=[[12.3313,-0.108707,416.142],
     [0.903899,-0.00320735,0.427666,0.00765917]];

EGMGetId egmID1;
EGMSetupAI ROB_1, egmID1 \Pose \aiR1x:=ai_01 \aiR2y:=ai_02
     \aiR3z:=ai_03 \aiR4rx:=ai_04 \aiR5ry:=ai_05 \aiR6rz:=ai_06;
EGMActPose egmID1 \Tool:=tool0 \WObj:=wobj0, posecor,
     EGM_FRAME_WOBJ, posesens, EGM_FRAME_TOOL \x:=egm_minmax_lin
     \y:=egm_minmax_lin \z:=egm_minmax_lin \rx:=egm_minmax_rot
     \ry:=egm_minmax_rot \rz:=egm_minmax_rot \LpFilter:=20;
EGMRunPose egmID1, EGM_STOP_HOLD \x \y \z \rx \ry \rz
     \RampInTime:=0.05;
```

V TRAP rutině:

```
EGMStop egmID1, EGM_STOP_HOLD;
```

Argumenty

```
EGMStop EGMid, Mode [\RampOutTime]
```

`EGMid`

Datový typ: `egmident`

identita EGM.

`Mode`

Datový typ: `egmstopmode`

Definuje způsob zakončení pohybu (`EGM_STOP_HOLD, EGM_STOP_RAMP_DOWN`)

`[\RampOutTime]`

Datový typ: `num`

Definuje v sekundách, jak rychle bude provedeno EGM rampa dolů.

Tento parametr nemá žádný význam, jestliže parametr `Mode` je nastaven na `EGM_STOP_HOLD`.

Omezení

- `EGMStop` může být použit pouze v pohybových úlohách RAPID.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
EGMStop
  [EGMid ':='] <variable (VAR) of egmident>','
  [Mode ':='] < expression (IN) of egmstopmode>
  ['\'RampOutTime ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.78 EOffsOff - Deaktivuje ofset pro pomocné osy

Použití

`EOffsOff` (*External Offset Off*) se používá pro deaktivaci ofsetu pro pomocné osy.

Ofset pro pomocné osy se aktivuje instrukcí `EOffsSet` nebo `EOffsOn` a vztahuje se na všechny pohyby, dokud není aktivován jiný ofset pro pomocné osy nebo dokud není ofset pro pomocné osy deaktivován.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `EOffsOff`:

Příklad 1

```
EOffsOff;
```

Deaktivace ofsetu pro pomocné osy.

Příklad 2

```
MoveL p10, v500, z10, tool1;
EOffsOn \ExeP:=p10, p11;
MoveL p20, v500, z10, tool1;
MoveL p30, v500, z10, tool1;
EOffsOff;
MoveL p40, v500, z10, tool1;
```

Ofset je definován jako rozdíl mezi pozicí každé osy na `p10` a `p11`. Tento posun ovlivňuje pohyb k `p20` a `p30`, ale nikoliv k `p40`.

Vykonávání programu

Aktivní ofsety pro pomocné osy jsou resetovány.

Syntaxe

```
EOffsOff ';'
```

Související informace

Pro informace o	Viz
Definice ofsetu pomocí dvou pozic	*EOffsOn - Aktivuje ofset pro pomocné osy na str 197*
Definice ofsetu pomocí známých hodnot	*EOffsSet - Aktivuje ofset pro pomocné osy pomocí známých hodnot na str 199*
Deaktivace posunu programu robotu	*PDispOff - Deaktivuje posun programu na str 476*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.79 EOffsOn - Aktivuje ofset pro pomocné osy

Použití

`EOffsOn` (*External Offset On*) se používá pro definování a aktivaci ofsetu pro pomocné osy pomocí dvou pozic.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `EOffsOn`:

Viz také *Další příklady na str 198*.

Příklad 1

```
MoveL p10, v500, z10, tool1;
EOffsOn \ExeP:=p10, p20;
```

Aktivace ofsetu pro pomocné osy. Vypočítá se pro každou osu na základě rozdílu mezi pozicemi `p10` a `p20`.

Příklad 2

```
MoveL p10, v500, fine \Inpos := inpos50, tool1;
EOffsOn *;
```

Aktivace ofsetu pro pomocné osy. Jelikož stop bod, který je přesně definován, byl použit v předchozí instrukci, argument `\ExeP` není nutné používat. Posun je vypočítán na základě rozdílu mezi aktuální pozicí každé osy a naprogramovaným bodem (*) uloženým v instrukci.

Argumenty

```
EOffsOn [\ExeP] ProgPoint
```

`[ \ExeP ]`

Executed Point

Datový typ: `robtarget`

Nová pozice použitá pro výpočet ofsetu. Jestliže je tento argument vypuštěn, použije se aktuální pozice os v době vykonávání programu.

`ProgPoint`

Programmed Point

Datový typ: `robtarget`

Původní pozice os v době programování.

Vykonávání programu

Ofset se vypočítává jako rozdíl mezi `\ExeP` a `ProgPoint` pro každou pomocnou osu. Jestliže nebylo určeno `\ExeP`, použije se místo toho aktuální pozice os v době vykonávání programu. Jelikož se používá skutečná pozice os, osy by se neměly při vykonávání `EOffsOn` pohybovat.

Tento ofset se potom použije na posun pozice pomocných os v následných polohovacích instrukcích a zůstane aktivní, dokud není některý jiný ofset aktivován

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.79 EOffsOn - Aktivuje ofset pro pomocné osy

RobotWare - OS

Pokračování

(instrukce `EOffsSet` nebo `EOffsOn`) nebo dokud není deaktivován ofset pro pomocné osy (instrukce `EOffsOff`).

Pouze **jeden** ofset pro každou individuální pomocnou osu může být aktivován ve stejnou dobu. Na druhé straně, několik `EOffsOn`, může být naprogramováno jeden po druhém, a pokud se tak stane, budou přidány odlišné ofsety.

Ofset pomocných os se automaticky resetuje:

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Další příklady

Více příkladů jak používat instrukci `EOffsOn` je názorně uvedeno dole.

Příklad 1

```
SearchL sen1, psearch, p10, v100, tool1;
PDispOn \ExeP:=psearch, *, tool1;
EOffsOn \ExeP:=psearch, *;
```

Provede se hledání a nalezená pozice robotu a pomocných os se uloží do pozice `psearch`. Každý pohyb, který je potom proveden, začíná od této pozice pomocí programového posunu robotu a pomocných os. To se vypočítává na základě rozdílu mezi vyhledanou pozicí a naprogramovaným bodem (*) uloženým v instrukci.

Syntaxe

```
EOffsOn
  [ '\' ExeP ':=' < expression (IN) of robtarget> ',']
  [ ProgPoint ':=' ] < expression (IN) of robtarget> ';'
```

Související informace

Pro informace o	Viz
Deaktivace ofsetu pro pomocné osy	*EOffsOff - Deaktivuje ofset pro pomocné osy na str 196*
Definice ofsetu pomocí známých hodnot	*EOffsSet - Aktivuje ofset pro pomocné osy pomocí známých hodnot na str 199*
Posun pohybů robotu	*PDispOn - Aktivuje posun programu na str 477*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.80 EOffsSet - Aktivuje ofset pro pomocné osy pomocí známých hodnot

Použití

`EOffsSet` (*External Offset Set*) se používá pro definování a aktivaci ofsetu pro pomocné osy pomocí známých hodnot.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `EOffsSet`:

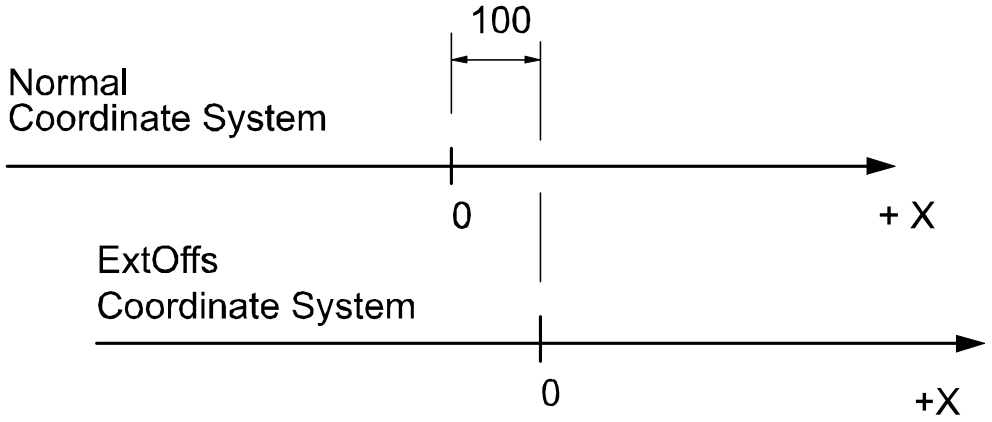
Příklad 1

```
VAR extjoint eax_a_p100 := [100, 0, 0, 0, 0, 0];
...
EOffsSet eax_a_p100;
```

Aktivace ofsetu `eax_a_p100` pro pomocné osy, znamená (za předpokladu, že logická pomocná osa „a“ je lineární) že:

- Souřadný systém `ExtOffs` je posunut o 100 mm pro logickou osu „a“ (viz obrázek dole).
- Dokud je tento ofset aktivní, všechny pozice budou posunuty o 100 mm ve směru osy x.

Následující obrázek ukazuje posun pomocné osy.



xx0500002162

Argumenty

```
EOffsSet EAxOffs
```

EAxOffs

External Axes Offset

Datový typ: `extjoint`

Ofset pro pomocné osy je definován jako data typu `extjoint`, vyjádřená v:

- mm pro lineární osy
- stupně pro otáčející se osy

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.80 EOffsSet - Aktivuje ofset pro pomocné osy pomocí známých hodnot

RobotWare - OS

Pokračování

Vykonávání programu

Ofset pro pomocné osy se aktivuje, když je vykonána instrukce `EOffsSet` a zůstává aktivní až do doby aktivace jiného ofsetu (instrukce `EOffsSet` nebo `EOffsOn`) nebo do deaktivace ofsetu pomocných os (instrukce `EOffsOff`).

Pouze jeden ofset pro pomocné osy může být aktivován ve stejném čase. Ofsety není možné k sobě přidávat pomocí `EOffsSet`.

Ofset pomocných os se automaticky resetuje:

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Syntaxe

```
EOffsSet
  [ EAxOffs ':=' ] < expression (IN) of extjoint> ';'
```

Související informace

Pro informace o	Viz
Aktivovat ofset pro pomocné osy	*EOffsOn - Aktivuje ofset pro pomocné osy na str 197*
Deaktivace ofsetu pro pomocné osy	*EOffsOff - Deaktivuje ofset pro pomocné osy na str 196*
Posun pohybů robotu	*PDispOn - Aktivuje posun programu na str 477*
Definice dat typu `extjoint`	*extjoint - Pozice externích svarů na str 1506*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.81 EraseModule - Vymazat modul

Použití

`EraseModule` se používá k odstranění modulu z paměti počítače během vykonávání.

Neexistují omezení v tom, jak byl modul načten. Mohl být načten ručně, z konfigurace nebo kombinací instrukcí `Load`, `StartLoad`, a `WaitLoad`.

Modul není možné definovat v konfiguraci jako *Shared*.

Základní příklady

Následující příklad názorně ukazuje instrukci `EraseModule`:

Příklad 1

```
EraseModule "PART_A";
```

Vymazat programový modul `PART_A` z paměti programu.

Argumenty

```
EraseModule ModuleName
```

`ModuleName`

Datový typ: `string`

Jméno modulu, který by měl být odstraněn. Všimněte si, že toto je jméno modulu, nikoliv jméno souboru.

Vykonávání programu

Vykonávání programu čeká na ukončení procesu odstranění programového modulu, potom vykonávání pokračuje další instrukcí.

Když je programový modul odstraněn, zbytek programových modulů bude propojen.

Omezení

Není dovoleno odstraňovat programový modul, který je ve fázi vykonávání.

TRAP rutiny, systémové I/O události a další programové úlohy se nemohou vykonávat během procesu odstraňování.

Vyloučit probíhající pohyby robotu během odstraňování.

Zastavení programu během instrukce `EraseModule` má za výsledek zastavení ochrany s vypnutím motorů a chybovou zprávou "20025 Stop order timeout" na FlexPendantu.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_MODULE`	Soubor v instrukci `EraseModule` nemůže být odstraněn, protože nebyl nalezen.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.81 EraseModule - Vymazat modul

RobotWare - OS

Pokračování

Syntaxe

```
EraseModule
[ModuleName':=']<expression (IN) of string>';'
```

Související informace

Pro informace o	Viz
Zrušit načtení programového modulu	*UnLoad - Stáhnout programový modul během provádění na str 905*
Načíst programový modul souběžně s vykonáváním jiného programu	*StartLoad - Načíst programový modul během vykonávání na str 703* *WaitLoad - Připojit načtený modul k úloze na str 947*
Přijmout nevyřešené reference	*Technická referenční příručka - Systémové parametry*, sekce *Controller*.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.82 ErrLog - Zapsat chybovou zprávu

Použití

`ErrLog` se používá k zobrazení chybové zprávy na FlexPendantu a její zapsání do protokolu událostí. Musí být uvedeno chybové číslo a pět chybových argumentů. Zpráva se uloží do procesní domény v protokolu robotu. Může se také použít `ErrLog` k zobrazení varování a informačních zpráv.

Základní příklady

Následující příklady názorně ukazují instrukci `ErrLog`:

Příklad 1

V případě, že nechcete vytvořit svůj vlastní soubor .xml, můžete použít `ErrorId 4800` jako v příkladu dole:

```
VAR errstr my_title := "myerror";
VAR errstr str1 := "errortext1";
VAR errstr str2 := "errortext2";
VAR errstr str3 := "errortext3";
VAR errstr str4 := "errortext4";
ErrLog 4800, my_title, str1,str2,str3,str4;
```

Na FlexPendantu bude zpráva vypadat takto:

Zpráva o události: 4800

myerror

errortext1

errortext2

errortext3

errortext4

Příklad 2

`ErrorId` musí být deklarován v souboru .xml. Číslo musí být mezi 5000 - 9999. Chybová zpráva se zapisuje do souboru .xml a argumenty ke zprávě jsou odeslány instrukcí `ErrLog`. `ErrorId` v souboru .xml je stejný, jako je stanoven v instrukci `ErrLog`.

POZNÁMKA: Při používání Errorld mezi 5000 - 9999 musíte nainstalovat svůj vlastní soubor .xml.

Příklad zprávy v souboru .xml:

```
<Message number="5210" eDefine="ERR_INPAR_RDONLY">
  <Title>Parameter error</Title>
  <Description>Task:<arg format="%s" ordinal="1" />
    <p />Symbol <arg format="%s" ordinal="2" />is read-only
    <p />Context:<arg format="%s" ordinal="3" /><p />
  </Description>
</Message>
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.82 ErrLog - Zapsat chybovou zprávu
RobotWare - OS
Pokračování

Příklad instrukce:

```
MODULE MyModule
  PROC main()
    VAR num errorid := 5210;
    VAR errstr arg := "P1";
    ErrLog errorid, ERRSTR_TASK, arg, ERRSTR_CONTEXT,ERRSTR_UNUSED,
         ERRSTR_UNUSED;
    ErrLog errorid \W, ERRSTR_TASK, arg,
         ERRSTR_CONTEXT,ERRSTR_UNUSED, ERRSTR_UNUSED;
  ENDPROC
ENDMODULE
```

Na FlexPendantu bude zpráva vypadat takto:

Zpráva o události: 5210

Chyba parametru

Úloha: T_ROB1

Symbol P1 je jen pro čtení.

Kontext: MyModule/main/ErrLog

První instrukce `ErrLog` generuje chybovou zprávu. Zpráva se uloží do protokolu robotu v procesní doméně. Je také zobrazena na FlexPendantu.

Druhá instrukce je varování. Zpráva se uloží pouze do protokolu robotu.

Programu bude pro provedení instrukce v obou případech pokračovat ve svém vykonávání.

Argumenty

```
ErrLog ErrorID [\W] | [\I] Argument1 Argument2 Argument3 Argument4
     Argument5
```

`ErrorId`

Datový typ: `num`

Číslo konkrétní chyby, která bude sledována. Číslo chyby musí být v intervalu 4800-4814, pokud se používá předinstalovaný soubor .xml, a mezi 5000-9999, při používání vlastního souboru .xml.

`[ \W ]`

Warning

Datový typ: `switch`

Vydává varování, které se ukládá pouze do protokolu událostí robotu (nezobrazuje se přímo na displeji FlexPendantu).

`[ \I ]`

Information

Datový typ: `switch`

Vydává informační zprávu, která se ukládá pouze do protokolu událostí (nezobrazuje se přímo na displeji FlexPendantu).

Jestliže žádný z argumentů `\W` nebo `\I` není určen, potom instrukce generuje chybovou zprávu přímo na FlexPendantu a zároveň ji uloží do protokolu událostí.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Argument1`

Datový typ: `errstr`

První argument v chybové zprávě. Může se použít jakýkoliv řetězec nebo předdefinová data typu `errstr`.

`Argument2`

Datový typ: `errstr`

Druhý argument v chybové zprávě. Může se použít jakýkoliv řetězec nebo předdefinová data typu `errstr`.

`Argument3`

Datový typ: `errstr`

Třetí argument v chybové zprávě. Může se použít jakýkoliv řetězec nebo předdefinová data typu `errstr`.

`Argument4`

Datový typ: `errstr`

Čtvrtý argument v chybové zprávě. Může se použít jakýkoliv řetězec nebo předdefinová data typu `errstr`.

`Argument5`

Datový typ: `errstr`

Pátý argument v chybové zprávě. Může se použít jakýkoliv řetězec nebo předdefinová data typu `errstr`.

Vykonávání programu

Na displeji jednotky FlexPendant se zobrazí chybová zpráva (max. 5 řádek) a poté je zapsána do protokolu událostí.

V případě argumentu `\W` nebo argumentu `\I` je do protokolu událostí zapsáno varování nebo informační zpráva.

`ErrLog` generuje programové chyby mezi 4800-4814 při používání souboru .xml instalovaného systémem, a mezi 5000-9999 při instalaci vlastního .xml souboru. Generovaná chyba závisí na indikovaném `ErrorID` .

Zpráva se ukládá do procesní domény v protokolu událostí.

Popis instalace vlastního souboru .xml je uveden v příručce *Další doplňky*, viz příslušné informace dole.

Omezení

Celková délka řetězce (Argument1-Argument5) je omezena na 195 znaků.

Syntaxe

```
ErrLog
  [ErrorId ':=' ] < expression (IN) of num> ','
  [ '\'W ] | [' \' I ] ','
  [Argument1 ':=' ] < expression (IN) of errstr> ','
  [Argument2 ':=' ] < expression (IN) of errstr> ','
  [Argument3 ':=' ] < expression (IN) of errstr> ','
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.82 ErrLog - Zapsat chybovou zprávu

RobotWare - OS

Pokračování

```
[Argument4 ':=' ] < expression (IN) of errstr> ','
[Argument5 ':=' ] < expression (IN) of errstr> ';'
```

Související informace

Pro informace o	Viz
Předdefinovaná data typu errstr	*errstr - Chybový řetězec na str 1502*
Zobrazit zprávu na jednotce FlexPendant	*TPWrite - Zapisuje na FlexPendant na str 793* *UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Protokol událostí	*Návod k použití - IRC5 s jednotkou FlexPendant*
Zprávy protokolu událostí, vysvětlení souboru .xml	*Příručka aplikace - Další doplňky*, sekce *Zprávy protokolu událostí*
Jak instalovat soubory XML při používání dalších doplňků	*Příručka aplikace - Další doplňky*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.83 ErrRaise - Zapisuje varování a volání chybového handleru

Použití

`ErrRaise` se používá pro vytvoření chyby v programu a potom volání chybového handleru rutiny. Varování se zapisuje do protokolu událostí. `ErrRaise` se může používat také v chybovém handleru pro šíření aktuální chyby k chybovému handleru volající rutiny.

Musí být uvedeno jméno chyby, číslo chyby a pět chybových argumentů. Zpráva se ukládá do procesní domény v protokolu robotu.

Základní příklady

Následující příklady názorně ukazují instrukci `ErrRaise`:

Příklad 1

V případě, že nechcete vytvořit svůj vlastní soubor .xml, můžete použít `ErrorId 4800` jako v příkladu dole:

```
MODULE MyModule
  VAR errnum ERR_BATT:=-1;
  PROC main()
    VAR num errorid := 4800;
    VAR errstr my_title := "Backup battery status";
    VAR errstr str1 := "Bacup battery is fully charged";
    BookErrNo ERR_BATT;
    ErrRaise "ERR_BATT", errorid, my_title, ERRSTR_TASK, str1,
         ERRSTR_CONTEXT,ERRSTR_EMPTY;
    ERROR
    IF ERRNO = ERR_BATT THEN
      TRYNEXT;
    ENDIF
  ENDPROC
ENDMODULE
```

Na FlexPendantu bude zpráva vypadat takto (varování a/nebo chyba):

Zpráva o události: 4800

Stav záložní baterie

Úloha: main

Záložní baterie je plně nabita

Kontext: MyModule/main/ErrRaise

Číslo chyby musí být rezervováno s instrukcí `BookErrNo`. Odpovídající řetězec je uveden jako první argument, `ErrorName` v `ErrRaise`.

`ErrRaise` vytváří chybu a potom volá chybový handler. Jestliže je o chybu postaráno, je generováno varování v protokolu událostí v procesní doméně. Jinak je generována fatální chyba a program se zastaví.

`ErrRaise` může být také použit v chybovém handleru v subrutině. V tomto případě pokračuje vykonávání v chybovém handleru volající rutiny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.83 ErrRaise - Zapisuje varování a volání chybového handleru
RobotWare - OS
Pokračování

Příklad 2

`ErrorId` musí být deklarován v souboru .xml. Číslo musí být mezi 5000 - 9999. Chybová zpráva se zapisuje do souboru .xml a argumenty ke zprávě jsou odeslány instrukcí `ErrRaise`. `ErrorId` v souboru .xml je stejný, jako je stanoven v instrukci `ErrRaise`.

POZNÁMKA: Při používání Errorld mezi 5000 - 9999 musíte nainstalovat svůj vlastní soubor .xml.

Příklad zprávy v souboru .xml:

```
<Message number="7055" eDefine="SYS_ERR_ARL_INPAR_RDONLY">
  <Title>Parameter error</Title>
  <Description>Task:<arg format="%s" ordinal="1" />
    <p />Symbol <arg format="%s" ordinal="2" />is read-only
    <p />Context:<arg format="%s" ordinal="3" /><p /></Description>
</Message>
```

Příklad instrukce:

```
MODULE MyModule
  VAR errnum ERR_BATT:=-1;
  PROC main()
    VAR num errorid := 7055;
    BookErrNo ERR_BATT;
    ErrRaise "ERR_BATT", errorid, ERRSTR_TASK,
         ERRSTR_CONTEXT,ERRSTR_UNUSED, ERRSTR_UNUSED,
         ERRSTR_UNUSED;
    ERROR
    IF ERRNO = ERR_BATT THEN
      TRYNEXT;
    ENDIF
  ENDPROC
ENDMODULE
```

Na FlexPendantu bude zpráva vypadat takto (varování a/nebo chyba):

Zpráva o události: 7055

Stav záložní baterie

Úloha: main

Záložní baterie je plně nabita

Kontext: MyModule/main/ErrRaise

Číslo chyby musí být rezervováno s instrukcí `BookErrNo`. Odpovídající řetězec je uveden jako první argument, `ErrorName` v `ErrRaise`.

`ErrRaise` vytváří chybu a potom volá chybový handler. Jestliže je o chybu postaráno, je generováno varování v protokolu událostí v procesní doméně. Jinak je generována fatální chyba a program se zastaví.

`ErrRaise` může být také použit v chybovém handleru v subrutině. V tomto případě pokračuje vykonávání v chybovém handleru volající rutiny.

Argumenty

```
ErrRaise ErrorName ErrorId Argument1 Argument2 Argument3 Argument4
     Argument5
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`ErrorName`

Datový typ: `string`

Číslo chyby se musí rezervovat pomocí instrukce `BookErrNo`. Odpovídající proměnná je uvedena jako `ErrorName`.

`ErrorId`

Datový typ: `num`

Číslo konkrétní chyby, která bude sledována. Číslo chyby musí být v intervalu 4800-4814, pokud se používá předinstalovaný soubor .xml, a mezi 5000-9999, při používání vlastního souboru .xml.

`Argument1`

Datový typ: `errstr`

První argument v chybové zprávě. Může se použít jakýkoliv řetězec nebo předdefinová data typu `errstr`.

`Argument2`

Datový typ: `errstr`

Druhý argument v chybové zprávě. Může se použít jakýkoliv řetězec nebo předdefinová data typu `errstr`.

`Argument3`

Datový typ: `errstr`

Třetí argument v chybové zprávě. Může se použít jakýkoliv řetězec nebo předdefinová data typu `errstr`.

`Argument4`

Datový typ: `errstr`

Čtvrtý argument v chybové zprávě. Může se použít jakýkoliv řetězec nebo předdefinová data typu `errstr`.

`Argument5`

Datový typ: `errstr`

Pátý argument v chybové zprávě. Může se použít jakýkoliv řetězec nebo předdefinová data typu `errstr`.

Vykonávání programu

`ErrRaise` generuje programová varování mezi 4800-4814 při používání souboru .xml instalovaného systémem, a mezi 5000-9999 při instalaci vlastního .xml souboru. Generovaná chyba závisí na indikovaném `ErrorID`. Varování se zapisuje do protokolu zpráv robotu v procesu domény.

Při vykonávání `ErrRaise` závisí chování na místě vykonávání:

- **Při vykonávání instrukce v těle rutiny je generováno varování a vykonávání pokračuje v chybovém handleru.**
- **Při vykonávání instrukce v chybovém handleru je staré varování přeskočeno, nové je generováno a ovládání je pozvednuto k volající instrukci.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.83 ErrRaise - Zapisuje varování a volání chybového handleru

RobotWare - OS

Pokračování

Omezení

Celková délka řetězce (Argument1-Argument5) je omezena na 195 znaků.

Další příklady

Více příkladů jak používat instrukci `ErrRaise` je názorně uvedeno dole.

Příklad 1

```
VAR errnum ERR_BATT:=-1;
VAR errnum ERR_NEW_ERR:=-1;

PROC main()
  testerrraise;
ENDPROC

PROC testerrraise()
  BookErrNo ERR_BATT;
  BookErrNo ERR_NEW_ERR;
  ErrRaise "ERR_BATT",7055,ERRSTR_TASK,ERRSTR_CONTEXT,
       ERRSTR_UNUSED,ERRSTR_UNUSED,ERRSTR_UNUSED;
  ERROR
  IF ERRNO = ERR_BATT THEN
    ErrRaise "ERR_NEW_ERR",7156,ERRSTR_TASK,ERRSTR_CONTEXT,
         ERRSTR_UNUSED,ERRSTR_UNUSED, ERRSTR_UNUSED;
  ENDIF
ENDPROC
```

Generovat nové varování 7156 z chybového handleru. Pozvednout ovladač k volající rutině a zastavit vykonávání.

Syntaxe

```
ErrRaise
  [ErrorName ':=' ] < expression (IN) of string> ','
  [ErrorId ':=' ] < expression (IN) of num> ','
  [Argument1 ':=' ] < expression (IN) of errstr> ','
  [Argument2 ':=' ] < expression (IN) of errstr> ','
  [Argument3 ':=' ] < expression (IN) of errstr> ','
  [Argument4 ':=' ] < expression (IN) of errstr> ','
  [Argument5 ':=' ] < expression (IN) of errstr> ';'
```

Související informace

Pro informace o	Viz
Předdefinovaná data typu errstr	*errstr - Chybový řetězec na str 1502*
Rezervování chybových čísel	*BookErrNo - Zapsat chybové číslo systému RAPID na str 41*
Řešení chyb	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.84 ErrWrite - Zapsat chybovou zprávu

Použití

`ErrWrite` (*Error Write*) se používá pro zobrazení chybové zprávy na FlexPendantu a její zapsání do protokolu událostí. Může se také používat pro zobrazení varování a informačních zpráv.

Základní příklady

Následující příklady názorně ukazují instrukci `ErrWrite`:

Příklad 1

```
ErrWrite "PLC error", "Fatal error in PLC" \RL2:="Call service";
Stop;
```

Zpráva se ukládá do protokolu robotu. Zpráva se také zobrazuje na displeji FlexPendantu.

Příklad 2

```
ErrWrite \W, "Search error", "No hit for the first search";
RAISE try_search_again;
```

Zpráva se ukládá pouze do protokolu robotu. Vykonávání programu potom pokračuje.

Argumenty

```
ErrWrite [ \W ] | [\I] Header Reason [ \RL2] [ \RL3] [ \RL4]
```

`[ \W ]`

Warning

Datový typ: `switch`

Vydává varování, které se ukládá pouze do protokolu chybových zpráv robotu (nezobrazuje se přímo na displeji FlexPendantu).

`[ \I ]`

Information

Datový typ: `switch`

Vydává informační zprávu, která se ukládá pouze do protokolu událostí (nezobrazuje se přímo na displeji FlexPendantu).

Jestliže žádný z argumentů `\W` nebo `\I` není určen, potom instrukce generuje chybovou zprávu přímo na FlexPendantu a zároveň ji uloží do protokolu událostí.

`Header`

Datový typ: `string`

Hlavička chybové zprávy (max. 46 znaků).

`Reason`

Datový typ: `string`

Příčina chyby.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.84 ErrWrite - Zapsat chybovou zprávu

RobotWare - OS

Pokračování

`[ \RL2]`

Reason Line 2

Datový typ: `string`

Příčina chyby.

`[ \RL3]`

Reason Line 3

Datový typ: `string`

Příčina chyby.

`[ \RL4]`

Reason Line 4

Datový typ: `string`

Příčina chyby.

Vykonávání programu

Na displeji jednotky FlexPendant se zobrazí chybová zpráva (max. 5 řádek) a poté je zapsána do protokolu zpráv robotu.

V případě argumentu `\W` nebo argumentu `\I` je do protokolu událostí zapsáno varování nebo informační zpráva.

`ErrWrite` generuje programovou chybu č. 80001 pro chybu, č. 80002 pro varování (`\W`) a č. 80003 pro informační zprávu (`\I`).

Omezení

Celková délka řetězce (Hlavička+Příčina+\RL2+\RL3+\RL4) je omezena na 195 znaků.

Syntaxe

```
ErrWrite
  [ '\'W ] | [ '\' I ] ','
  [ Header ':=' ] < expression (IN) of string>','
  [ Reason ':=' ] < expression (IN) of string>
  [ ’\’RL2 ':=' < expression (IN) of string> ]
  [ ’\’RL3 ':=' < expression (IN) of string> ]
  [ ’\’RL4 ':=' < expression (IN) of string> ] ';'
```

Související informace

Pro informace o	Viz
Předdefinovaná data typu errstr	*errstr - Chybový řetězec na str 1502*
Zobrazit zprávu na jednotce FlexPendant	*TPWrite - Zapisuje na FlexPendant na str 793* *UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Protokol událostí	*Návod k použití - IRC5 s jednotkou FlexPendant*
Zapsat chybovou zprávu - Err Log	*ErrLog - Zapsat chybovou zprávu na str 203*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.85 EXIT - Ukončuje vykonávání programu

Použití

`EXIT` se používá pro ukončení vykonávání programu. Restart programu bude potom blokován, to znamená, že program může být restartován pouze od první instrukce hlavní rutiny.

Instrukce `EXIT` by se měla používat při vzniku fatální chyby nebo když vykonávání programu musí být zastaveno natrvalo. Instrukce `Stop` se používá pro dočasné zastavení vykonávání programu. Po vykonání instrukce `EXIT` ukazatel programu zmizí. Aby vykonávání programu mohlo pokračovat, ukazatel programu musí být nastaven.

Základní příklady

Následující příklad názorně ukazuje instrukci `EXIT`:

Příklad 1

```
ErrWrite "Fatal error","Illegal state";
EXIT;
```

Vykonávání programu se zastaví a nemůže být restartováno od této pozice v programu.

Syntaxe

```
EXIT ';'
```

Související informace

Pro informace o	Viz
Dočasné zastavení vykonávání programu	*Stop - Ukončuje vykonávání programu na str 731*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.86 ExitCycle - Přerušit aktuální cyklus a začít nový

Použití

`ExitCycle` se používá k přerušení aktuální cyklu a posunutí ukazatele programu (PP) na první instrukci v hlavní rutině.

Jestli program je vykonáván v nepřetržitém režimu, začne vykonávat další cyklus.

Jestliže vykonávání je v cyklickém režimu, vykonávání se zastaví u první instrukce v hlavní rutině.

Základní příklady

Následující příklad názorně ukazuje instrukci `ExitCycle`:

Příklad 1

```
VAR num cyclecount:=0;
VAR intnum error_intno;

PROC main()
  IF cyclecount = 0 THEN
    CONNECT error_intno WITH error_trap;
    ISignalDI di_error,1,error_intno;
  ENDIF
  cyclecount:=cyclecount+1;
  ! start to do something intelligent
  ...
ENDPROC

TRAP error_trap
  TPWrite "ERROR, I will start on the next item";
  ExitCycle;
ENDTRAP
```

Tím se spustí další cyklus, jestliže je nastaven signál `di_error`.

Vykonávání programu

Provedení `ExitCycle` v programové úloze, která řídí mechanické jednotky, má za výsledek následující v aktuální úloze:

- Probíhající pohyby robotu se zastaví.
- Všechny dráhy robotu, které nejsou prováděny na všech úrovních dráhy (normální a `StorePath` úroveň), jsou vynulovány.
- Všechny instrukce, které jsou spuštěny, ale nejsou dokončeny na všech úrovních vykonávání (normální a `TRAP` úroveň), jsou přerušeny.
- Ukazatel programu je přesunut k první instrukci v hlavní rutině.
- Vykonávání programu pokračuje k dalšímu cyklu.

Provedení `ExitCycle` v některé jiné programové úloze, která neřídí mechanické jednotky, má za výsledek následující v aktuální úloze:

- Všechny instrukce, které jsou spuštěny, ale nejsou dokončeny na všech úrovních vykonávání (normální a `TRAP` úroveň), jsou přerušeny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

- Ukazatel programu je přesunut k první instrukci v hlavní rutině.
- Vykonávání programu pokračuje k dalšímu cyklu.

Všechny ostatní modální záležitosti v programu a systému **nejsou** ovlivněny `ExitCycle`, jako je:

- Skutečná hodnota proměnných nebo perzistentů.
- Všechna pohybová nastavení jako sekvence `StorePath-RestoPath` , světové zóny a tak dále.
- Otevřít soubory, adresáře atd.
- Definovaná přerušení atd.

Když se používá `ExitCycle` ve voláních rutiny a vstupní rutina je definována s “Move PP to Routine ...” nebo “Call Routine ...”,`ExitCycle` přeruší aktuální cyklus a posune ukazatel programu zpět k první instrukci ve vstupní rutině (namísto hlavní rutiny, jak bylo určeno předtím).

Syntaxe

```
ExitCycle';'
```

Související informace

Pro informace o	Viz
Zastavení po fatální chybě	*EXIT - Ukončuje vykonávání programu na str 213*
Ukončení provádění programu	*EXIT - Ukončuje vykonávání programu na str 213*
Zastavení pro činnosti programu	*Stop - Ukončuje vykonávání programu na str 731*
Ukončení vykonávání rutiny	*RETURN - Ukončení vykonávání rutiny na str 549*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.87 FOR - Opakuje, kolikrát je stanoveno

Použití

FOR se používá, když jedna nebo několik instrukcí mají být několikrát opakovány.

Základní příklady

Následující příklady názorně ukazují instrukci FOR:

Viz také *Další příklady na str 216*.

Příklad 1

```
FOR i FROM 1 TO 10 DO
  routine1;
ENDFOR
```

Opakuje proceduru routine1 10-krát.

Argumenty

```
FOR Loop counter FROM Start value TO End value [STEP Step value]
     DO ... ENDFOR
```

`Loop counter`

Identifier

Jméno dat, která budou obsahovat hodnotu aktuálního počítadla smyček. Data jsou deklarována automaticky.

Jestliže jméno počítadla smyček je stejné jako jakákoliv data, která již existují v aktuálním rámci, existující data budou skryta ve smyčce FOR a nebudou nijak ovlivněna.

`Start value`

Datový typ: Num

Požadovaná spouštěcí hodnota počítadla smyček. (obvykle hodnoty celého čísla)

`End value`

Datový typ: Num

Požadovaná koncová hodnota počítadla smyček. (obvykle hodnoty celého čísla)

`Step value`

Datový typ: Num

Hodnota, o kterou bude počítadlo smyček přirůstat (nebo se snižovat) při každé smyčce. (obvykle hodnoty celého čísla)

Jestliže tato hodnota není určena, hodnota kroku bude automaticky nastavena na 1 (nebo -1, jestliže spouštěcí hodnota je větší než koncová hodnota).

Další příklady

Více příkladů jak používat instrukci FOR je názorně uvedeno dole.

Příklad 1

```
FOR i FROM 10 TO 2 STEP -2 DO
  a{i} := a{i-1};
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
ENDFOR
```

Hodnoty v poli se upravují směrem nahoru, takže `a{10}:=a{9}`, `a{8}:=a{7}` a tak dále.

Vykonávání programu

1. Výrazy pro start, konec a krokové hodnoty se vyhodnocují.
2. Počítadlu smyček je přidělena spouštěcí (start) hodnota.
3. Hodnota počítadla smyček se kontroluje, aby bylo vidět, jestli jeho hodnota leží mezi spouštěcí a koncovou hodnotou nebo jestli je totožná se spouštěcí nebo koncovou hodnotou. Jestliže hodnota počítadla smyček je mimo toto pásmo, smyčka `FOR` se zastaví a vykonávání programu pokračuje s instrukcí následující `ENDFOR`.
4. Instrukce ve smyčce `FOR` jsou provedeny.
5. Počítadlo smyček je zvýšeno (nebo sníženo) v souladu s krokovou hodnotou.
6. Smyčka `FOR` se opakuje od bodu 3.

Omezení

K počítadlu smyček (datového typu `num`) je možné přistoupit ze smyčky `FOR` a následně skrývá jiná data a rutiny, které mají stejné jméno. Může být pouze čteno (nikoliv aktualizováno) instrukcemi ve smyčce `FOR`.

Desetinné hodnoty pro start, konec nebo stop v kombinaci s přesnými podmínkami ukončení pro smyčku `FOR` se nemohou používat (nedefinováno, jestli poslední smyčka běží nebo nikoliv).

Poznámky

Jestliže počet opakování by měl být opakován dokud je daný výraz vyhodnocován k hodnotě `TRUE`, instrukce `WHILE` by měla být použita místo toho.

Syntaxe

```
FOR <loop variable> FROM <expression> TO <expression>
  [ STEP <expression> ] DO
  <statement list>
ENDFOR
```

Související informace

Pro informace o	Viz
Výrazy	*Technická referenční příručka - Přehled RAPID*
Opakuje se, dokud...	*WHILE - Opakuje se, dokud... na str 976*
Identifikátory	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.88 FricIdInit - Spustit identifikaci tření

Použití

`FricIdInit` označuje počáteční bod sekvence pohybových instrukcí, které budou opakovány kvůli výpočtu interního tření robotu.

Příklad

Základní příklad, kde se používá kruhový pohyb pro výpočet interního tření robotu pro tento pohyb:

```
PERS num friction_levels{6};

! Start of the friction calculation sequence
FricIdInit;

! Execute the move sequence
MoveC p10, p20, Speed, z0, Tool;
MoveC p30, p40, Speed, z0, Tool;

! Repeat the sequence and calculate the friction
FricIdEvaluate friction_levels;

! Activate compensation for the calculated friction levels
FricIdSetFricLevels friction_levels;
```

Požadavky

Systémový parametr *Friction FFW On* se musí nastavit na TRUE. Jinak nebude instrukce `FricIdInit` dělat nic.

Omezení

- `FricIdInit` funguje pouze pro TCP roboty.
- Může být provedeno pouze z pohybových úloh.
- Robot se musí posunout na úroveň základní dráhy.
- Ladění tření se nemůže kombinovat se synchronizovaným pohybem. To znamená, že `SyncMoveOn` není povolen mezi `FricIdInit` a `FricIdEvaluate`.
- Pohybová sekvence, kvůli které se ladění tření provádí, musí začínat a končit s jemným bodem. Pokud to tak není, jemné body budou automaticky vloženy během procesu ladění.

Syntaxe

```
FricIdInit ';'
```

Související informace

Pro informace o	Viz
Advanced robot motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.89 FricIdEvaluate - Vyhodnotit identifikaci tření

Použití

`FricIdEvaluate` přinutí robot opakovat pohyb mezi dvěma instrukcemi `FricIdInit` a `FricIdEvaluate`, zatímco bude vypočítávat tření pro každou osu robotu.

Příklad

Základní příklad, kde se používá kruhový pohyb pro výpočet interního tření robotu pro tento pohyb:

```
PERS num friction_levels{6};

! Start of the friction calculation sequence
FricIdInit;

! Execute the move sequence
MoveC p10, p20, Speed, z0, Tool;
MoveC p30, p40, Speed, z0, Tool;

! Repeat the sequence and calculate the friction
FricIdEvaluate friction_levels;

! Activate compensation for the calculated friction levels
FricIdSetFricLevels friction_levels;
```

Argumenty

```
FricIdEvaluate FricLevels [\MechUnit] [\BwdSpeed] [\NoPrint]
     [\FricLevelMax] [\FricLevelMin] [\OptTolerance]
```

`FricLevels`

Friction levels

Datový typ: `array of num`

Když je dokončeno `FricIdEvaluate`, pole `FricLevels` bude obsahovat laděné úrovně tření pro všechny osy robotu. Toto pole musí být deklarováno, aby mělo stejný počet počet prvků jako má robot os. Pamatujte, že instrukce `FricIdSetFricLevels` musí být volána, aby tyto hodnoty měly účinek.

`[\MechUnit]`

Mechanical unit

Datový typ: `mecunit`

Argument `MechUnit` je volitelný. Jestliže je vypuštěn, ladění tření bude provedeno pro mechanickou jednotku zastoupenou předdefinovanou RAPID proměnnou `ROB_ID`, která je referencí k TCP robotu v aktuální programové úloze. Kompenzace tření je možná jen u robotů TCP.

`[\BwdSpeed]`

Backward speed

Datový typ: `speeddata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.89 FricIdEvaluate - Vyhodnotit identifikaci tření
RobotWare - OS
Pokračování

Po každé iteraci v procesu ladění se pohyb posune zpět podél naprogramované dráhy. Podle implicitní hodnoty je pohyb zpět proveden naprogramovanou rychlostí. Pro urychlení procesu se může použít volitelný argument `BwdSpeed` kvůli určení vyšší rychlosti během pohybu zpět. To *neovlivní* výsledek ladění.

`[\NoPrint]`

Datový typ: `switch`

Jestliže se použije argument `NoPrint`, není zapsán žádný text na FlexPendant o postupu iterací identifikace tření.

`[\FricLevelMax]`

Friction level max

Datový typ: `num`

Normálně se optimální hodnota tření nalezne zkoušením hodnot mezi 1 % a 500 % konfigurované hodnoty tření. Ve vzácných případech může toto vyvolat chybovou zprávu (Chyba rychlosti spoje). Abyste tomu předešli, použijte argument `FricLevelMax` a nastavte ho na hodnotu nižší než 500. Například, jestliže je `FricLevelMax` nastaven na 400, budou testovány hodnoty mezi 1 % a 400 %

Přípustné jsou hodnoty 101-500.

`[\FricLevelMin]`

Friction level min

Datový typ: `num`

Normálně se optimální hodnota tření nalezne zkoušením hodnot mezi 1 % a 500 % konfigurované hodnoty tření. Pro nastavení vyšší počáteční hodnoty než 1 % použijte argument `FricLevelMin`. Například, jestliže je `FricLevelMin` nastaven na 80, budou testovány hodnoty mezi 80 % a 500 %.

Přípustné jsou hodnoty 1-99.

`[\OptTolerance]`

Optimization tolerance

Datový typ: `num`

Normálně se optimální hodnota tření nalezne zkoušením hodnot až do dosažení malé tolerance. Pro urychlení procesu může být tato hodnota zvýšena. Zvýšení této hodnoty může přinést méně přesný výsledek.

Přípustné hodnoty jsou 1-10. Výchozí hodnota je 1.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_FRICTUNE_FATAL`	Během ladění tření se objeví chyba.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

POZNÁMKA

Jestliže ladění tření nemá žádný efekt, zkontrolujte, jestli systémový parametr *Friction FFW On* je nastaven na TRUE.

Požadavky

Systémový parametr *Friction FFW On* se musí nastavit na TRUE. Jinak nebude instrukce `FricIdEvaluate` dělat nic.

Omezení

- `FricIdEvaluate` funguje pouze pro TCP roboty.
- `FricIdEvaluate` může být vykonáno pouze z pohybových úloh.
- Robot se musí posunout na úroveň základní dráhy.
- U systému MultiMove se může ladění tření provádět pouze pro jeden robot současně. Několik robotů může provádět `FricIdEvaluate`současně, ale budou automaticky odstaveny a budou čekat na své pořadí, dokud bude jiný robot zaměstnán vykonáváním laděním tření.
- Ladění tření se nemůže kombinovat se synchronizovaným pohybem. To znamená, že `SyncMoveOn` není povolen mezi `FricIdInit` a `FricIdEvaluate`.
- Pohybová sekvence, kvůli které se ladění tření provádí, musí začínat a končit s jemným bodem. Pokud to tak není, jemné body budou automaticky vloženy během procesu ladění.

Syntaxe

```
FricIdEvaluate
  [ FricLevels ':='] < persistent array {*} (PERS) of num >
  ['\' MechUnit ':=' < variable (VAR) of mecunit >]
  ['\' BwdSpeed ':=' < expression (IN) of speeddata >]
  ['\' NoPrint]
  ['\' FricLevelMax ':=' < expression (VAR) of num >]
  ['\' FricLevelMin ':=' < expression (VAR) of num >]
  ['\' OptTolerance ':=' < expression (VAR) of num >] ';'
```

Související informace

Pro informace o	Viz
Advanced robot motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.90 FricIdSetFricLevels - Nastavit úrovně tření po identifikaci tření

Použití

`FricIdSetFricLevels` se používá pro nastavení úrovně tření pro každou osu mechanické jednotky.

Příklad

Základní příklad, kde se používá kruhový pohyb pro výpočet interního tření robotu pro tento pohyb:

```
PERS num friction_levels{6};

! Start of the friction calculation sequence
FricIdInit;

! Execute the move sequence
MoveC p10, p20, Speed, z0, Tool;
MoveC p30, p40, Speed, z0, Tool;

! Repeat the sequence and calculate the friction
FricIdEvaluate friction_levels;

! Activate compensation for the calculated friction levels
FricIdSetFricLevels friction_levels;
```

Argumenty

```
FricIdSetFricLevels FricLevels [\MechUnit]
```

`FricLevels`

Friction levels

Datový typ: `array of num`

Pole `FricLevels` určuje úroveň tření pro každou osu v procentu výchozí hodnoty tření. Hodnoty musí být v intervalu 0-500.

`[\MechUnit]`

Mechanical unit

Datový typ: `mecunit`

Argument `MechUnit` je volitelný. Jestliže je vypuštěn, hodnoty tření budou nastaveny pro mechanickou jednotku zastoupenou předdefinovanou RAPID proměnnou `ROB_ID`. Kompenzace tření je možná jen u robotů TCP.

Vykonávání programu

Nastavení úrovní tření zůstane aktivní až do:

- Vykonávání programu je spuštěno od začátku (PP na Main)
- Bylo provedeno další volání k `FricIdSetFricLevels`
- Nový program byl načten
- Ovladač je restartován pomocí restartovacího režimu **Resetovat systém**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Požadavky

Systémový parametr *Friction FFW On* se musí nastavit na TRUE. Jinak nebude instrukce `FricIdSetFricLevels` dělat nic.

Omezení

- `FricIdSetFricLevels` funguje pouze pro TCP roboty.

Syntaxe

```
FricIdSetFricLevels
  [ FricLevels ':='] < array {*} (IN) of num >
  ['\' MechUnit ':=' < variable (VAR) of mecunit >] ';'
```

Související informace

Pro informace o	Viz
Advanced robot motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.91 GetDataVal - Získat hodnotu datového objektu

Použití

`GetDataVal` (*Get Data Value*) umožňuje získat hodnotu od datového objektu, který je určen s proměnnou řetězce.

Základní příklady

Následující příklady názorně ukazují instrukci `GetDataVal`:

Příklad 1

```
VAR num value;
...
GetDataVal "reg"+ValToStr(ReadNum(mycom)),value;
```

Tím se získá hodnota registru s číslem, které je přijato od sériového kanálu `mycom`. Hodnota bude uložena v proměnné `value`.

Příklad 2

```
VAR datapos block;
VAR string name;
VAR num valuevar;
...
SetDataSearch "num" \Object:="my.*" \InMod:="mymod";
WHILE GetNextSym(name,block) DO
  GetDataVal name\Block:=block,valuevar;
  TPWrite name+" "\Num:=valuevar;
ENDWHILE
```

Tato akce vytiskne k FlexPendantu všechny proměnné `num`, které začínají s `my` v modulu `mymod` s jeho hodnotou.

Příklad 3

```
VAR num NumArrConst_copy{2};
...
GetDataVal "NumArrConst", NumArrConst_copy;
TPWrite "Pos1 = " \Num:=NumArrConst_copy{1};
TPWrite "Pos2 = " \Num:=NumArrConst_copy{2};
```

Tato akce vytiskne proměnné `num` v poli `NumArrConst`.

Argumenty

```
GetDataVal Object [\Block]|[\TaskRef]|[\TaskName] Value
```

`Object`

Datový typ: `string`

Jméno datového objektu.

`[ \Block ]`

Datový typ: `datapos`

Blok vložený k datovému objektu. Toto může být získáno pouze s funkcí `GetNextSym`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je tento argument vypuštěn, bude získána hodnota viditelného datového objektu v rámci současného vykonávání programu.

`[\TaskRef]`

Task Reference

Datový typ: `taskid`

Identita programové úlohy, ve které se má hledat určený datový objekt. Při použití tohoto argumentu můžete hledat deklarace `PERS` nebo `TASKPERS` v jiných úlohách, všechny ostatní deklarace povedou k chybě.

Pro všechny programové úlohy v systému budou dostupné předdefinované proměnné datového typu `taskid`. Identita proměnné bude například "taskname"+"Id", pro úlohu `T_ROB1` bude identita proměnné `T_ROB1Id`.

`[\TaskName]`

Datový typ: `string`

Jméno programové úlohy, ve které se má hledat určený datový objekt. Při použití tohoto argumentu můžete hledat deklarace `PERS` nebo `TASKPERS` v jiných úlohách, všechny ostatní deklarace povedou k chybě.

`Value`

Datový typ: `anytype`

Proměnná pro uložení hodnoty get. Datový typ musí být stejný, jako je datový typ pro datový objekt, který se hledá. Hodnota get se může získat z konstanty, proměnné nebo perzistentu, ale musí být uložena do proměnné.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_SYM_ACCESS`	• Datový objekt neexistuje. • Datový objekt jsou data rutiny nebo parametr rutiny a není umístěn v aktuální aktivní rutině. • Hledání v jiných úlohách jiných deklarací než `PERS` nebo `TASK PERS`.
`ERR_INVDIM`	Datový objekt a proměnná použité v argumentu `Value` mají odlišné rozměry.
`ERR_SYMBOL_TYPE`	Datový objekt a proměnná použité v argumentu `Value` jsou odlišného typu. Při používání datových typů `ALIAS` dostanete také tuto CHYBU, i když typy mohou mít stejný základní datový typ.

Při používání argumentů `TaskRef` nebo `TaskName` můžete hledat deklarace `PERS` nebo`TASK PERS` v jiných úlohách, všechny ostatní deklarace budou mít za výsledek chybu a systémová proměnná `ERRNO` se nastaví na `ERR_SYM_ACCESS`. Hledání `PERS` deklarovaného jako `LOCAL` v jiných úlohách bude mít za výsledek také chybu a systémová proměnná `ERRNO` se nastaví na `ERR_SYM_ACCESS`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.91 GetDataVal - Získat hodnotu datového objektu

RobotWare - OS

Pokračování

Omezení

U polohodnotového datového typu není možné hledat datový typ s propojenou hodnotou. Například, při hledání `dionum`, nedostanete žádný nález pro signály `signaldi`, a jestliže budete hledat `num`, nedostanete žádný nález pro signály `signalgi` ani `signalai`.

Není možné získat hodnotu proměnné deklarované jako `LOCAL` ve vestavěném modulu RAPID.

Syntaxe

```
GetDataVal
  [ Object ':=' ] < expression (IN) of string >
  ['\'Block' :='<variable (VAR) of datapos>]
  |[ '\'TaskRef' :=' <variable (VAR) of taskid>]
  |[ '\'TaskName' :=' <expression (IN) of string>] ',']
  [ Value ':=' ] <variable (VAR) of anytype>]';'
```

Související informace

Pro informace o	Viz
Definovat sadu symbolů ve vyhledávací akci	*SetDataSearch - Definovat sadu symbolů ve vyhledávací sekvenci na str 622*
Získat další odpovídající symbol	*GetNextSym - Získat další odpovídající symbol na str 1175*
Nastavit hodnotu datového objektu	*SetDataVal - Nastavit hodnotu datového objektu na str 626*
Nastavit hodnotu mnoha datových objektů	*SetAllDataVal - Nastavit hodnotu všech datovým objektům v definované sadě na str 618*
Související datový typ datapos	*datapos - Přiložený blok pro datový objekt na str 1482*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.92 GetSysData - Získat systémová data

Použití

`GetSysData` získává hodnotu a (doplňkově) jméno symbolu pro aktuální systémová data určeného typu.

S touto instrukcí je možné získat data a jméno aktuálního aktivního nástroje, pracovního objektu, užitečné zátěže nebo celkové zátěže pro robot ve skutečné nebo připojené pohybové úloze, nebo jakékoliv jmenovité pohybové úloze.

Základní příklady

Následující příklady názorně ukazují instrukci `GetSysData`:

Příklad 1

```
PERS tooldata curtoolvalue := [TRUE, [[0, 0, 0], [1, 0, 0, 0]],
     [2, [0, 0, 2], [1, 0, 0, 0], 0, 0, 0]];
VAR string curtoolname;
GetSysData curtoolvalue;
```

Kopírovat aktuální hodnotu dat aktivního nástroje do proměnné perzistentu `curtoolvalue`.

Příklad 2

```
GetSysData curtoolvalue \ObjectName := curtoolname;
```

Kopírovat také aktuální jméno aktivního nástroje do proměnné `curtoolname`.

Příklad 3

```
PERS loaddata curload;
PERS loaddata piece:=[2.8,[-38.2,-10.1,-73.6],[1,0,0,0],0,0,0];
PERS loaddata
     tool2piece:=[13.1,[104.5,13.5,115.9],[1,0,0,0],0,0,0.143];
PERS tooldata tool2 := [TRUE, [[138.695,150.023,98.9783],
     [0.709396,-0.704707,-0.00856676,0.00851007]],
     [10,[105.2,-3.8,118.7], [1,0,0,0],0,0,0.123]];
VAR string name;
..
IF GetModalPayloadMode() = 1 THEN
  GripLoad piece;
  MoveL p3, v1000, fine, tool2;
  ..
  ..
  ! Get current payload
  GetSysData curload \ObjectName := name;
ELSE
  MoveL p30, v1000, fine, tool2\TLoad:=tool2piece;
  ..
  ..
  ! Get current total load
  GetSysData curload \ObjectName := name;
ENDIF
```

Jestliže `ModalPayLoadMode` je 1, zkopírujte aktuální aktivní užitečnou zátěž a jméno do proměnné `name`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.92 GetSysData - Získat systémová data
RobotWare - OS
Pokračování

Jestliže `ModalPayLoadMode` je 0, zkopírujte aktuální celkovou zátěž a jméno do proměnné `name`.

Argumenty

```
GetSysData [\TaskRef]|[\TaskName] DestObject[\ObjectName ]
```

`[\TaskRef]`

Task Reference

Datový typ: `taskid`

Identita programové úlohy, ze které by měla být načtena data aktuálně aktivních systémových dat.

Pro všechny programové úlohy v systému budou dostupné předdefinované proměnné datového typu `taskid`. Identita proměnné bude například "taskname"+"Id", pro úlohu `T_ROB1` bude identita proměnné `T_ROB1Id`.

`[\TaskName]`

Datový typ: `string`

Jméno programové úlohy, ze které by měla být načtena aktuálně aktivní systémová data.

Jestliže žádný z argumentů `\TaskRef` or `\TaskName` není určen, potom se použije aktuální úloha.

`DestObject`

Datový typ: `anytype`

Proměnná perzistentu pro uložení hodnoty aktuálně aktivních systémových dat.

Datový typ tohoto argumentu také určuje typ systémových dat (nástroj, pracovní objekt nebo užitečná zátěž / celková zátěž) pro získání. Při použití volitelného argumentu TLoad u pohybových instrukcí je získána celková zátěž namísto užitečné zátěže, jestliže je použit datový typ loaddata.

Datový typ	Typ systémových dat
tooldata	Nástroj
wobjdata	Pracovní objekt
loaddata	Užitečná zátěž/Celková zátěž

Pole nebo komponent záznamu se nemohou používat.

`[\ObjectName]`

Datový typ:`string`

Volitelný argument (proměnná nebo perzistent) pro získání jména aktuálně aktivních systémových dat.

Vykonávání programu

Při provádění instrukce `GetSysData` je hodnota aktuálních dat uložena do určené proměnné perzistentu v argumentu `DestObject`.

Jestliže se použije argument `\ObjectName`, jméno aktuálních dat se uloží do určené proměnné nebo perzistentu v argumentu `ObjectName`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Aktuální systémová data pro nástroj, pracovní objekt nebo celkovou zátěž se aktivují vykonáním kterékoliv pohybové instrukce. Užitečná zátěž se aktivuje vykonáním instrukce `GripLoad`.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_NOT_MOVETASK`	Argumenty `\TaskRef` nebo `\TaskName` určují nepohybovou úlohu. **POZNÁMKA** Žádná chyba nebude generována, jestliže argumenty `\TaskRef` nebo `\TaskName` určují nepohybovou úlohu, která vykonává tuto funkci `GetSysData` (reference k mé vlastní nepohybové úloze). Aktuální systémová data budou potom získána od připojené pohybové úlohy.

Syntaxe

```
GetSysData
  ['\' TaskRef ':=' <variable (VAR) of taskid>]
  |['\' TaskName' :=' <expression (IN) of string>]
  [ DestObject' :='] < persistent(PERS) of anytype>
  ['\'ObjectName' :=' < variable or persistent (INOUT) of string>
       ] ';'
```

Související informace

Pro informace o	Viz
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice užitečné zátěže	*loaddata - Zátěžová data na str 1523*
Nastavit systémová data	*SetSysData - Nastavit systémová data na str 634*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.93 GetTrapData - Získat data přerušení pro aktuální TRAP

Použití

`GetTrapData` se používá v trap rutině, aby byly získány všechny informace o přerušení, které způsobilo vykonání trap rutiny.

Používá se v trap rutinách generovaných instrukcí `IError`, před použitím instrukce `ReadErrData`.

Základní příklady

Následující příklad názorně ukazuje instrukci `GetTrapData`:

Viz také *Další příklady na str 230*.

Příklad 1

```
VAR trapdata err_data;
GetTrapData err_data;
```

Uložit informaci přerušení do nehodnotové proměnné `err_data`.

Argumenty

```
GetTrapData TrapEvent
```

`TrapEvent`

Datový typ: `trapdata`

Proměnná pro uložení informací o tom, co způsobilo vykonání trapu.

Omezení

Tuto instrukci je možné použít pouze v rutině `TRAP`.

Další příklady

Více příkladů instrukce `GetTrapData` je názorně uvedeno dole.

Příklad 1

```
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
VAR trapdata err_data;
...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
ENDTRAP
```

Jestliže chyba je zachycena do trap rutiny `trap_err`, chybová doména, chybové číslo a typ chyby jsou uloženy do příslušných nehodnotových proměnných typu `trapdata`.

Syntaxe

```
GetTrapData
  [TrapEvent ':='] <variable (VAR) of trapdata>';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*
Více informací o správě přerušení	*Technická referenční příručka - Přehled RAPID*
Data přerušení pro aktuální TRAP	*trapdata - Data přerušení pro aktuální TRAP na str 1617*
Přikazuje přerušení na chyby	*IError - Přikazuje přerušení na chyby na str 246*
Získává informace o chybě	*ReadErrData - Získává informace o chybě na str 527*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.94 GOTO - Přechází na novou instrukci

Použití

`GOTO` se používá k přenosu vykonávání programu na jinou řádku (návěstí) v rámci stejné rutiny.

Základní příklady

Následující příklad názorně ukazuje instrukci `GOTO`:

Příklad 1

```
GOTO next;
...
next:
```

Vykonávání programu pokračuje s následující instrukcí.

Příklad 2

```
reg1 := 1;
next:
...
reg1 := reg1 + 1;
IF reg1<=5 GOTO next;
```

Vykonávání bude přeneseno na `next` čtyřikrát (pro `reg1`= 2, 3, 4, 5).

Příklad 3

```
IF reg1>100 THEN
  GOTO highvalue
ELSE
  GOTO lowvalue
ENDIF
lowvalue:
...
GOTO ready;
highvalue:
...
ready:
```

Jestliže `reg1` je větší než `100`, vykonávání bude přeneseno k návěstí `highvalue`, jinak bude vykonávání přeneseno k návěstí `lowvalue`.

Argumenty

```
GOTO Label
```

`Label`

Identifier

Návěstí, od kterého bude pokračovat vykonávání programu.

Omezení

Je možné pouze přenést vykonávání programu na návěstí v rámci stejné rutiny.

Je možné pouze přenést vykonávání programu na návěstí v rámci instrukce `IF` nebo `TEST`, jestliže instrukce `GOTO` je také umístěna ve stejném skoku této instrukce.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Je možné pouze přenést vykonávání programu na návěstí v rámci instrukce `FOR` nebo `WHILE`, jestliže instrukce `GOTO` je také umístěna v této instrukci.

Syntaxe

```
GOTO <identifier>';'
```

Související informace

Pro informace o	Viz
Štítek	*Návěstí - Jméno řádky na str 327*
Jiné instrukce, které mění tok programu	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.95 GripLoad - Definuje užitečnou zátěž pro robot

Použití

`GripLoad` se používá k definování užitečné zátěže, kterou robot drží ve svém chapadlu.

Popis

`GripLoad` určuje, kterou zátěž robot nese. Určená zátěž se používá k nastavení dynamického modelu robotu tak, aby pohyby robotu bylo možné řídit nejlepším možným způsobem.

Užitečná zátěž se připojuje/odpojuje pomocí instrukce `GripLoad`, která přičítá nebo odečítá váhu užitečné zátěže k váze chapadla.

VAROVÁNÍ

Je důležité vždy definovat skutečné zatížení nástroje a pokud se používá, užitečné zatížení robota (např. uchopený kus). Nesprávné definice zátěžových dat mohou vést k přetížení mechanické konstrukce robota.

Uvedení nesprávných údajů může často vést k následujícím důsledkům:

- Nebude využita maximální kapacita robota
- Bude narušena přesnost dráhy s rizikem minutí
- Vznikne riziko přetížení mechanické konstrukce

Základní příklady

Následující příklady ilustrující instrukci `GripLoad`, jsou uvedeny dole.

Příklad 1

```
Set gripper;
WaitTime 0.3;
GripLoad piece1;
```

Připojení užitečné zátěže, `piece1`, určeno ve stejný čas, když robot uchopí zátěž.

Příklad 2

```
Reset gripper;
WaitTime 0.3;
GripLoad load0;
```

Odpojení užitečné zátěže, určeno ve stejný čas, když robot odloží užitečnou zátěž.

Argumenty

```
GripLoad Load
```

`Load`

Datový typ: `loaddata`

Data zátěže, která popisují aktuální užitečnou zátěž.

Je možné nechat program běžet zkušebně bez užitečné zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je digitální vstupní signál nastaven na 1, `loaddata` v instrukci `GripLoad` není uvažován a použije se pouze `loaddata` v aktuálním `tooldata`.

Vykonávání programu

Určená zátěž ovlivňuje výkon robotu.

Je automaticky nastavena výchozí zátěž (`load0`), 0 kg

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Užitečná zátěž se aktualizuje pro mechanickou jednotku, která je řízena aktuální programovou úlohou. Jestliže se použije `GripLoad` z nepohybové úlohy, užitečná zátěž se aktualizuje pro mechanickou jednotku řízenou připojenou pohybovou úlohou.

Syntaxe

```
GripLoad
  [ Load ':=' ] < persistent (PERS) of loaddata > ';'
```

Související informace

Pro informace o	Viz
Zátěžová identifikace zátěže nástroje, užitečné zátěže nebo zátěže ramena	*Návod k použití - IRC5 s jednotkou FlexPendant*, sekce *Programování a testování - Servisní rutiny*
Definovat užitečnou zátěž pro mechanické jednotky	*MechUnitLoad - Definuje užitečnou zátěž pro mechanickou jednotku na str 343*
Definice zátěžových dat	*loaddata - Zátěžová data na str 1523*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.96 HollowWristReset - Resetovat duté zápěstí pro IRB5402 a IRB5403

Použití

`HollowWristReset` (***Reset hollow wris*t**) resetuje pozici spojů zápěstí na manipulátorech dutých zápěstí, jako je IRB5402 a IRB5403.

Instrukce umožňuje zabránit přetočení spojů zápěstí 4 a 5, když byly otočeny o jednu nebo více otáček. Po vykonání instrukce `HollowWristReset` mohou spoje zápěstí pokračovat v otáčení stejným směrem.

Popis

*`HollowWristReset` usnadňuje dělat aplikační programy. Nemusíte zajišťovat, aby pozice zápěstí byla v rámci ±2 otáček v době programování a může se ušetřit čas cyklu, protože robot nebude ztrácet čas přetáčením zápěstí. Je zde omezení ±144 otáček pro otočení spojů 4 a 5, než je pozice zápěstí resetována pomocí `HollowWristReset`. Programátor robotu musí počítat s tímto omezením a brát ho do úvahy při plánování robotických programů. Abyste zajistili, že limit 144 otáček nebude překročen po provedení programu „otáčení zápěstí“ několikrát za sebou, měli byste vždy nechat robot přijít k celkovému zastavení a resetovat absolutní pozici v každém programu (nebo cyklu/rutině/modulu a tak dále podle potřeby). Všimněte si, že všechny osy musí zůstat zastavené během provádění instrukce `HollowWristReset` . Jakmile jsou tato omezení vzata do úvahy, spoje 4 a 5 se mohou během vykonávání programu otáčet neurčitě dlouho a nezávisle na spoji 6.

Používejte `HollowWristReset` namísto `IndReset` pro resetování dutého zápěstí, protože tato instrukce udržuje limity spoje pro spoj 6, aby bylo zabráněno přílišnému kroucení nátěrových trubek/kabelů.

Základní příklady

Následující příklad názorně ukazuje instrukci `HollowWristReset`:

Příklad 1

```
MoveL p10,v800,fine,paintgun1\WObj:=workobject1;
HollowWristReset;
```

Všechny aktivní osy jsou zastaveny u stop bodu a zápěstí je resetováno.

Omezení

Všechny aktivní osy musí být zastaveny při vykonávání instrukce `HollowWristReset`.

Spoje zápěstí musí být resetovány předtím, než kterýkoliv z nich dosáhne limitu ±144 otáček (51840st/904rad).

Kdykoliv se objeví zastavení programu, nouzové zastavení, zastavení při selhání napájení atd., ovladač udržuje kontext dráhy, aby byl schopen vrátit se do dráhy a nechat robot pokračovat ve vykonávání programu od bodu na dráze, kde došlo k zastavení. V ručním režimu, jestliže manipulátor byl odsunut ven z dráhy mezi zastavením a restartem, operátor je informován následující zprávou na FlexPendantu: „**Mimo dráhu! Robot byl posunut po zastavení programu. Má se**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

robot vrátit na dráhu při spuštění? Ano/Ne/Zrušit”. To poskytuje možnost vrátit se na dráhu před restartem. V automatickém režimu se robot automaticky vrací na dráhu.

`HollowWristReset` **odstraňuje kontext dráhy. To znamená, že není možné vrátit se na dráhu v případě restartu programu, jestliže byla mezitím provedena instrukce** `HollowWristReset`**. Jestliže je tato instrukce vykonána ručně (“Debug + Call Routine...” v editoru programu), měla by být vykonána pouze v době, kdy není návrat na dráhu požadován. To znamená, po úplném dokončení programu nebo po úplném dokončení instrukce v krokovém vykonávání a manipulátor není vysunut ven z dráhy krokováním (jogging) atd.**

Syntaxe

```
HollowWristReset ´;’
```

Související informace

Pro informace o	Viz
Související systémové parametry	*Technická referenční příručka - Systémové parametry*
Vrátit se na dráhu	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.97 ICap - připojit události CAP k trap rutinám

Použití

`ICap` se používá ke spojení čísla přerušení (které je již připojeno k trap rutině) se specifickou událostí CAP, viz argumenty dole a seznam dostupných událostí. Při používání `ICap` se vytvoří spojení mezi konkrétní procesní událostí a uživatelsky vytvořenou trap rutinou. Jinými slovy, zmíněná trap rutina se vykoná, když se objeví spojená událost CAP.

Doporučujeme umístit trapy do úlohy v pozadí.

Základní příklad

Dole je příklad, kde událost CAP CAP_START je spojena s trap rutinou `start_trap`.

```
VAR intnum start_intno:=0;
...

TRAP start_trap
  ! This routine will be executed when the event CAP_START is
       reported from the core
  ! Do what you want to do
ENDTRAP

PROC main()
  IDelete start_intno;
  CONNECT start_intno WITH start_trap;
  ICap start_intno, CAP_START;
  CapL p1, v100, cdata, weavestart, weave, z50, gun1;
ENDPROC
```

Argumenty

```
ICap Interrupt Event
```

Interrupt

Datový typ: `intnum`

Identita přerušení. Předtím by mělo dojít k připojení k trap rutině prostřednictvím instrukce `CONNECT`.

Event

Datový typ: `num`

Číslo události CAP bude spojeno s přerušením. Tyto události jsou předdefinované konstanty.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Dostupné události CAP

Viz sekce *Vazba mezi fázemi a událostmi* v *Application manual - Continuous Application Platform*.

Události	Fáze	Číslo události	Popis
CAP_START		0	Tato událost se objeví, jakmile je spuštěn proces CAP. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
START_PRE	PRE	1	Tato událost se objeví, když je aktivován (pokud je přítomen) dohled fáze PRE. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
PRE_STARTED	PRE	2	Tato událost se objeví, když jsou splněny všechny požadavky na seznamu dohledů PRE, to znamená, když je spuštěna fáze PRE_START. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
START_MAIN	START	3	Tato událost vzniká, když fáze PRE_START je ukončena a fáze MAIN je spuštěna.
MAIN_STARTED	START	4	Tato událost vzniká, když jsou splněny všechny podmínky seznamu dohledů START, to znamená, když je spuštěna fáze MAIN.
STOP_WEAVESTART	MAIN	5	Tato událost se objeví před každým startem weave - ale jen když je start weave přikázán. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
WEAVESTART_REGAIN	MAIN	6	Tato událost se objeví, když robot se vrátil zpět ke dráze po startu weave Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
MOTION_DELAY	MAIN	7	Tato událost se objeví po prodlevě (pokud existuje) spuštění pohybu. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
STARTSPEED_TIME	MAIN	8	Tato událost vznikne, když uběhne čas pro použití *Rychlosti startu* a je čas pro přepnutí na hlavní pohybová data.
MAIN_MOTION	MAIN	9	Tato událost vznikne, když je hlavní motor aktivován při běžícím procesu.
MOVE_STARTED	MAIN	10	Tato událost se objeví, jakmile robot zahájí pohyb podél procesní dráhy. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
RESTART	MAIN	11	Tato událost vznikne, když je přikázán restart.
NEW_INSTR	MAIN	12	Tato událost vznikne, když nová instrukce `CapL` nebo `CapC` je získána z programu RAPID.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.97 ICap - připojit události CAP k trap rutinám
Continuous Application Platform (CAP)
Pokračování

Události	Fáze	Číslo události	Popis
AT_POINT	MAIN	13	Tato událost vznikne u každého robtarget na procesní dráze, kromě bodu startu a konce.
AT_RESTART-POINT	MAIN	14	Tato událost se objeví, když robot krokoval zpět (jog) na vzdálenost restartu, na procesní dráze po zastavení.
LAST_SEG-MENT	MAIN	15	Tato událost se objeví v bodě startu posledního segmentu.
PRO-CESS_END_POINT	MAIN	16	Tato událost se objeví, když robot dosáhne koncového bodu procesu, to znamená, kde je podpora procesu pro ukončení. Při použití *flying end* není distribuována žádná událost.
END_MAIN	END_MAIN	17	Tato událost se objeví v bodě na procesní dráze, kde je spuštěn dohled koncové sekvence, to znamená, když robot dosáhne koncového bodu procesu.
MAIN_ENDED	END_MAIN	18	Tato událost vzniká, když jsou splněny všechny podmínky seznamu dohledů END_MAIN, to znamená, když je hlavní proces považován na ukončený.
PATH_END_POINT		19	Tato událost vznikne, když robot dosáhne koncového bodu dráhy, tj. jemného bodu nebo středu zóny (u *flying end*) v poslední instrukci CAP.
PRO-CESS_EN-DED		20	Tato událost vznikne, když proces je ukončen na jemném bodu nebo ve středu zóny (u *flying end*) v poslední instrukci CAP.
END_POST1	END_POST1	21	Tato událost vznikne, když je čas ukončit fázi POST1, tj. když je čas provést změnu od fáze POST1 k fázi POST2. Při použití *flying end* není distribuována žádná událost.
POST1_EN-DED	END_POST1	22	Tato událost nastane, když jsou splněny všechny podmínky seznamu dohledů END_POST1, tj. když je úspěšně ukončena fáze POST1 a fáze POST2 je spuštěna. Při použití *flying end* není distribuována žádná událost.
END_POST2	END_POST2	23	Tato událost vznikne, když je fáze POST2 na svém konci, tj. když je čas ukončit proces. Při použití *flying end* není distribuována žádná událost.
POST2_EN-DED	END_POST2	24	Tato událost nastane, když jsou splněny všechny podmínky seznamu dohledů END_POST2, tj. když je úspěšně ukončena fáze POST2 a tím i celý proces. Při použití *flying end* není distribuována žádná událost.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Události	Fáze	Číslo události	Popis
CAP_STOP		25	Tato událost je požadovanou událostí. Jestliže se používá jakákoliv jiná událost, musí být také definována. Událost/trap se vykonána co nejdříve po zastavení ovladače kvůli chybě nebo zastavení programu. Chyba může být odstranitelná, detekovaná v CAP, fatální chyba detekovaná v CAP nebo interní chyba zastavující ovladač. Kód vykonávaný v tomto trapu by měl přivést veškeré externí vybavení do bezpečného stavu, například resetovat všechny externí I/O signály.
CAP_PF_RE-START	MAIN	26	Tato událost vznikne, když je přikázán restart.
EQUIDIST	MAIN	27	Tato událost je odeslána, jestliže je přikázána instrukcí `CapEquiDist`.
AT_ERROR-POINT	MAIN	28	Tato událost se objevuje po restartu, když TCP dosáhne pozice chyby dohledu.
FLY_START	MAIN	29	Tato událost se objeví při používání *flying start*.. Událost je dostupná pouze s *flying start*.
FLY_END	MAIN	30	Tato událost se objeví při používání *flying end*.. Událost je dostupná pouze s *flying end*.
LAST_INSTR_EN-DED	MAIN	31	Tato událost se objeví, když vykonávání RAPID poslední instrukce CAP je dokončeno během *flying end*. Událost je dostupná pouze s *flying end*.
END_PRE	PRE	32	Tato událost se objeví, když je aktivován (pokud je přítomen) dohled fáze PRE. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
PRE_ENDED	PRE	33	Tato událost se objeví, když je aktivován (pokud je přítomen) dohled fáze PRE. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
START_POST1	POST1	34	Tato událost se objeví, když je aktivován (pokud je přítomen) dohled fáze POST1. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
POST1_STAR-TED	POST1	35	Tato událost se objeví, když je aktivován (pokud je přítomen) dohled fáze POST1. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.
START_POST2	POST2	36	Tato událost se objeví, když je aktivován (pokud je přítomen) dohled fáze POST1. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.97 ICap - připojit události CAP k trap rutinám

Continuous Application Platform (CAP)

Pokračování

Události	Fáze	Číslo události	Popis
POST2_STARTED	POST2	37	Tato událost se objeví, když je aktivován (pokud je přítomen) dohled fáze POST1. Při použití *flying start* není distribuována žádná událost, protože zde je již pohyb TCP. Při restartu je tato událost distribuována.

Omezení

Stejná proměnná pro identitu přerušení se nemůže použít více než jednou, aniž by ta první byla vymazána. Přerušení by se tady měla ošetřovat tak, jak je ukázáno na jedné z alternativ dole.

```
PROC setup_events ()
  VAR intnum start_intno;
  IDelete start_intno;
  CONNECT start_intno WITH start_trap;
  ICap start_intno, CAP_START;
ENDPROC
```

Veškerá aktivace přerušení se provádí na začátku programu. Tyto instrukce jsou potom uchovány vně hlavního toku programu. Instrukce `ICap` by měla být vykonána pouze jednou, například od rutiny událostí spouštěcího systému. Doporučení je takové, že trapy by měly být umístěny v úloze v pozadí.

Syntaxe

```
ICap
  [Interrupt ':='] < variable (IN) of intnum > ','
  [Event ':='] < variable (IN) of num > ';'
```

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Spojit přerušení s trapem	*CONNECT - Připojuje přerušení k trap rutině na str 133*
Zrušit přerušení připojené k trapu	*IDelete - Zruší přerušení na str 243*
Datový typ `intnum`	*intnum - Identita přerušení na str 1516*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.98 IDelete - Zruší přerušení

Použití

`IDelete` (*Vymazání přerušení*) se používá ke zrušení (vymazání) subskripce přerušení.

Jestliže přerušení má být jen dočasně vypnuto, měla by se použít instrukce `ISleep` nebo `IDisable`

Základní příklady

Následující příklad názorně ukazuje instrukci `IDelete`:

Příklad 1

```
IDelete feeder_low;
```

Přerušení `feeder_low` je zrušeno.

Argumenty

```
IDelete Interrupt
```

`Interrupt`

Datový typ: `intnum`

Identita přerušení.

Vykonávání programu

Definice přerušení byla zcela vymazána. Aby bylo možné ho znovu definovat, nejprve musí být připojeno k trap rutině.

Doporučuje se dát stop bod před `IDelete` . Jinak bude přerušení deaktivováno ještě před dosažením koncového bodu dráhy pohybu.

Přerušení se nemusí mazat; to se stane automaticky, když

- nový program byl načten
- program byl restartován od začátku
- ukazatel programu byl přesunut na začátek rutiny

Syntaxe

```
IDelete [ Interrupt ´:=’ ] < variable (VAR) of intnum > ´;’
```

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*
Více informací o správě přerušení	*Technická referenční příručka - Přehled RAPID*
Dočasné vypnutí přerušení	*ISleep - Deaktivuje přerušení na str 318*
Dočasné vypnutí všech přerušení	*IDisable - Vypíná přerušení na str 244*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.99 IDisable - Vypíná přerušení

Použití

`IDisable`*(Vypnutí přerušení)* se používá pro vypnutí všech přerušení dočasně. Může se to používat například v konkrétní citlivé části programu, kde nesmějí být povolena žádná přerušení, jestliže ruší normální vykonávání programu.

Základní příklady

Následující příklad názorně ukazuje instrukci `IDisable`:

Příklad 1

```
IDisable;
FOR i FROM 1 TO 100 DO
  character[i]:=ReadBin(sensor);
ENDFOR
IEnable;
```

Nejsou povolena žádná přerušení, dokud sériový kanál čte.

Vykonávání programu

Přerušení, která se objeví během času, kdy probíhá instrukce `IDisable`, jsou umístěna do fronty. Když jsou přerušení povolena ještě jednou, potom okamžitě začínají s generováním a jsou vykonávána ve frontě v pořadí „první dovnitř - první ven“.

`IEnable` je aktivní jako standard. `IEnable` je automaticky nastaveno

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno
- po vykonání jednoho cyklu (přechod `main`) nebo vykonání `ExitCycle`.

Syntaxe

```
IDisable´;’
```

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*
Více informací o správě přerušení	*Technická referenční příručka - Přehled RAPID*
Povolování přerušení	*IEnable - Zapíná přerušení na str 245*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.100 IEnable - Zapíná přerušení

Použití

`IEnable`(*Zapnutí přerušení*) se používá pro zapnutí přerušení během vykonávání programu.

Základní příklady

Následující příklad názorně ukazuje instrukci `IEnable`:

Příklad 1

```
IDisable;
FOR i FROM 1 TO 100 DO
  character[i]:=ReadBin(sensor);
ENDFOR
IEnable;
```

Žádná přerušení nejsou povolena, když sériový kanál čte. Když ukončí čtení, přerušení jsou ještě jednou povolena.

Vykonávání programu

Přerušení, která se objeví během času, kdy probíhá instrukce `IDisable`, jsou umístěna do fronty. Když jsou přerušení povolena ještě jednou (`IEnable`), přerušení potom okamžitě začnou s generováním a jsou vykonána ve frontě systémem „první dovnitř - první ven“. Vykonávání programu potom pokračuje běžným způsobem a přerušení, která se objeví později, jsou ošetřena, jakmile se objeví.

Přerušení jsou povolena vždy, když program je spuštěn od začátku. Přerušení vypnutá instrukcí `ISleep` nejsou ovlivněna instrukcí `IEnable`.

Syntaxe

```
IEnable´;'
```

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*
Více informací o správě přerušení	*Technická referenční příručka - Přehled RAPID*
Nepovolování žádných přerušení	*IDisable - Vypíná přerušení na str 244*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.101 IError - Přikazuje přerušení na chyby

Použití

`IError` (*Interrupt Errors*) se používá pro přikazování a zapínání přerušení, když se objeví chyba.

Chyba, varování nebo změna stavu mohou být zapsány s `IError`.

Základní příklady

Následující příklad názorně ukazuje instrukci `IError`:

Viz také *Další příklady na str 247*.

Příklad 1

```
VAR intnum err_int;
...
PROC main()
  CONNECT err_int WITH err_trap;
  IError COMMON_ERR, TYPE_ALL, err_int;
```

Přikazuje přerušení v RAPIDu a vykonání `TRAP` rutiny `err_trap` pokaždé, když je v systému generována chyba, varování nebo změna stavu.

Argumenty

```
IError ErrorDomain [\ErrorId] ErrorType Interrupt
```

`ErrorDomain`

Datový typ: `errdomain`

Chybová doména, která bude sledována. Viz předdefinovaná data typu `errdomain`. Pro určení jakékoliv domény použijte `COMMON_ERR`.

`[ \ErrorId ]`

Datový typ: `num`

Volitelně, číslo konkrétní chyby, která bude sledována. Číslo chyby musí být uvedeno bez první číslice (chybová doména) kompletního čísla chyby.

Například, 10008 Program restartován, musí být uvedeno jako 0008 nebo jen 8.

`ErrorType`

Datový typ: `errtype`

Typ události, jako je chyba, varování nebo změna stavu, která bude sledována. Viz předdefinovaná data typu `errtype`. Pro určení jakéhokoliv typu použijte `TYPE_ALL`.

`Interrupt`

Datový typ: `intnum`

Identita přerušení. Předtím by mělo dojít k připojení k trap rutině prostřednictvím instrukce `CONNECT`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Odpovídající trap rutina je automaticky volána, když se objeví chyba v určené doméně určeného typu a volitelně s určeným číslem chyby. Když je toto vykonáno, pokračuje vykonávání programu od místa, kde přerušení vzniklo.

Další příklady

Více příkladů instrukce `IError` je názorně uvedeno dole.

```
VAR intnum err_interrupt;
VAR trapdata err_data;
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
PROC main()
  CONNECT err_interrupt WITH trap_err;
  IError COMMON_ERR, TYPE_ERR, err_interrupt;
  ...
  IDelete err_interrupt;
  ...
ENDPROC
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
  ! Set domain no 1 ... 11
  SetGO go_err1, err_domain;
  ! Set error no 1 ...9999
  SetGO go_err2, err_number;
ENDTRAP
```

Když se objeví chyba (pouze chyba, nikoliv varování nebo změna stavu) číslo chyby je vyhledáno v trap rutině a tato hodnota se použije k nastavení 2 skupin digitálních výstupních signálů.

Omezení

Není možné přikazovat přerušení na interní chyby.

V úloze typu `NORMAL` bude událost zahozena během zastavení programu, takže nikoliv všechny události mohou být získány v úloze NORMAL. Pro získání všech událostí musí být úloha typu statická nebo polostatická.

Stejná proměnná pro identitu přerušení se nemůže použít více než jednou, aniž by ta první byla vymazána. Přerušení by se tady měla ošetřovat tak, jak je ukázáno na jedné z alternativ dole.

```
VAR intnum err_interrupt;
PROC main ( )
  CONNECT err_interrupt WITH err_trap;
  IError COMMON_ERR, TYPE_ERR, err_interupt;
  WHILE TRUE DO
  :
  :
  ENDWHILE
ENDPROC
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.101 IError - Přikazuje přerušení na chyby

RobotWare - OS

Pokračování

Přerušení jsou aktivována na začátku programu. Tyto instrukce na začátku jsou potom držena mimo hlavní tok programu.

```
VAR intnum err_interrupt;
PROC main ( )
  CONNECT err_interrupt WITH err_trap;
  IError COMMON_ERR, TYPE_ERR, err_interupt;
  :
  :
  IDelete err_interrupt;
ENDPROC
```

Přerušení je vymazáno na konci programu a je potom znovu aktivováno. Všimněte si v tomto případě, že přerušení je po krátkou dobu neaktivní.

Syntaxe

```
IError
  [ErrorDomain ':='] <expression (IN) of errdomain>
  ['\'ErrorId':=' <expression (IN) of num>\\ ','
  [ErrorType' :='] <expression (IN) of errtype> ´,'
  [Interrupt' :='] <variable (VAR) of intnum>';'
```

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*
Více informací o správě přerušení	*Technická referenční příručka - Přehled RAPID*
Chybové domény, předdefinované konstanty	*errdomain - Chybová doména na str 1493*
Typy chyb, předdefinované konstanty	*errtype - Typ chyby na str 1503*
Získat data přerušení pro aktuální TRAP	*GetTrapData - Získat data přerušení pro aktuální TRAP na str 230*
Získává informace o chybě	*ReadErrData - Získává informace o chybě na str 527*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.102 IF - Jestliže je splněna podmínka, potom ...; jinak ...

Použití

`IF` se používá, když mají být vykonány různé instrukce v závislosti na tom, jestli je splněna podmínka nebo nikoliv.

Základní příklady

Základní příklady instrukce `IF` jsou názorně uvedeny dole.

Viz také *Další příklady na str 249*.

Příklad 1

```
IF reg1 > 5 THEN
  Set do1;
  Set do2;
ENDIF
```

Signály `do1` a `do2` se nastavují pouze když `reg1` je větší než `5`.

Příklad 2

```
IF reg1 > 5 THEN
  Set do1;
  Set do2;
ELSE
  Reset do1;
  Reset do2;
ENDIF
```

Signály `do1` a `do2` se nastavují nebo resetují podle toho, jestli `reg1` je větší než `5` nebo nikoliv.

Argumenty

```
IF Condition THEN ...
  {ELSEIF Condition THEN ...}
[ELSE ...]
ENDIF
```

`Condition`

Datový typ: `bool`

Podmínka, která musí být splněna, aby mohly být vykonány instrukce mezi `THEN` a `ELSE/ELSEIF`.

Další příklady

Více příkladů jak používat instrukci `IF` je názorně uvedeno dole.

Příklad 1

```
IF counter > 100 THEN
  counter := 100;
ELSEIF counter < 0 THEN
  counter := 0;
ELSE
  counter := counter + 1;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.102 IF - Jestliže je splněna podmínka, potom ...; jinak ...
RobotWare - OS
Pokračování

```
ENDIF
```

`counter` je zvyšován o `1`. Nicméně, jestliže hodnota `counter` je mimo limit `0-100`, `counter` dostane přidělenu odpovídající limitní hodnotu.

Vykonávání programu

Podmínky se testují v postupném pořadí, dokud jedna z nich není splněna. Vykonávání programu pokračuje s instrukcemi spojenými s touto podmínkou. Jestliže není splněna žádná podmínka, vykonávání programu pokračuje s instrukcemi následujícími `ELSE`. Jestliže je splněna více než jedna podmínka, pouze instrukce spojené s první z těchto podmínek budou vykonány.

Syntaxe

```
IF <conditional expression> THEN
  <statement list>
{ ELSEIF <conditional expression> THEN
  <statement list> | <EIT> }
[ ELSE
  <statement list> ]
ENDIF
```

Související informace

Pro informace o	Viz
Podmínky (logické výrazy)	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.103 Incr - Přírůstky po 1

Použití

`Incr` se používá pro přičtení 1 k numerické proměnné nebo perzistentu.

Základní příklady

Následující příklad názorně ukazuje instrukci `Incr`:

Viz také *Další příklady na str 251*.

Příklad 1

```
Incr reg1;
```

1 je připočítáno k `reg1`, tj. `reg1:=reg1+1`.

Argumenty

```
Incr Name | Dname
```

`Name`

Datový typ: `num`

Jméno proměnné nebo perzistentu, které budou změněny.

`Dname`

Datový typ: `dnum`

Jméno proměnné nebo perzistentu, které budou změněny.

Další příklady

Více příkladů instrukce `Incr` je názorně uvedeno dole.

Příklad 1

```
VAR num no_of_parts:=0;
...
WHILE stop_production=0 DO
  produce_part;
  Incr no_of_parts;
  TPWrite "No of produced parts= "\Num:=no_of_parts;
ENDWHILE
```

Počet vyrobených kusů je aktualizován po každém cyklu na FlexPendantu. Produkce pokračuje v běhu, dokud vstupní signál `stop_production` není nastaven.

Příklad 2

```
VAR dnum no_of_parts:=0;
...
WHILE stop_production=0 DO
  produce_part;
  Incr no_of_parts;
  TPWrite "No of produced parts= "\Dnum:=no_of_parts;
ENDWHILE
```

Počet vyrobených kusů je aktualizován po každém cyklu na FlexPendantu. Produkce pokračuje v běhu, dokud vstupní signál `stop_production` není nastaven.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.103 Incr - Přírůstky po 1

RobotWare - OS

Pokračování

Syntaxe

```
Incr
  [ Name ':=' ] < var or pers (INOUT) of num >
  | [ Dname' :=' ] < var or pers (INOUT) of dnum >' ;'
```

Související informace

Pro informace o	Viz
Snižování proměnné o 1	*Decr - Snižování po 1 na str 152*
Přidání jakékoliv hodnoty k proměnné	*Add - Přidává numerickou hodnotu na str 26*
Změna dat pomocí libovolného výrazu, například násobení	*":=" - Přiděluje hodnotu na str 33*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.104 IndAMove - Nezávislý pohyb absolutní pozice

Použití

`IndAMove`(*Independent Absolute Movement*) se používá pro změnu osy do nezávislého režimu a posunutí osy do určité pozice.

Nezávislá osa je taková osa, která se pohybuje nezávisle na ostatních osách v robotickém systému. Jelikož vykonávání programu pokračuje okamžitě, je možné vykonat jiné instrukce (včetně polohovacích instrukcí) během času, kdy se nezávislá osa pohybuje.

Jestliže osa bude posunuta s otáčkou, měla by se místo toho použít instrukce `IndRMove`. Jestliže pohyb bude proveden na krátkou vzdálenost od aktuální pozice, musí se použít instrukce `IndDMove`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `IndAMove`:

Viz také *Další příklady na str 255*.

Příklad 1

```
IndAMove Station_A,2\ToAbsPos:=p4,20;
```

Osa `2` od `Station_A` se posune do pozice `p4` rychlostí `20` stupňů/s.

Argumenty

```
IndAMove MecUnit Axis [\ToAbsPos] | [\ToAbsNum] Speed [\Ramp]
```

`MecUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`Axis`

Datový typ: `num`

Číslo aktuální osy pro mechanickou jednotku (1-6)

`[\ToAbsPos]`

To Absolute Position

Datový typ: `robtarget`

Pozice osy určená jako `robtarget`. Použije se pouze komponent pro tuto konkrétní `Axis`. Hodnota se použije jako hodnota absolutní pozice ve stupních (mm u lineárních os).

Pozice osy bude ovlivněna, jestliže osa je posunuta pomocí instrukce `EOffsSet` or `EOffsOn`.

Pro osy robotu se místo toho použije argument `\ToAbsNum`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.104 IndAMove - Nezávislý pohyb absolutní pozice

Independent Axis
Pokračování

`[\ToAbsNum]`

To Absolute Numeric value

Datový typ: `num`

Pozice osy definovaná ve stupních (mm u lineární osy).

Pomocí tohoto argumentu NEBUDE pozice ovlivněna žádným posuvem, například `EOffsSet` nebo `PDispOn`.

Stejná funkce jako `\ToAbsPos`, ale pozice je definována jako numerická hodnota kvůli usnadnění ruční změny pozice.

`Speed`

Datový typ: `num`

Rychlost osy ve stupních (mm/s u lineární osy).

`[\Ramp]`

Datový typ: `num`

Snížit zrychlení a zpomalení od max výkonu (1-100%,100%=maximumperformance).

Vykonávání programu

Když je vykonáván `IndAMove`, určená osa se pohybuje naprogramovanou rychlostí k určené pozici osy. Jestliže je naprogramován `\Ramp`, dojde ke snížení zrychlení/zpomalení.

Pro změnu osy zpět do normálního režimu se použije instrukce `IndReset`. Ve spojení s tím může být změněna logická pozice osy tak, že počet otáček je z pozice vymazán, například, aby se zabránilo otáčení zpět k dalšímu pohybu.

Rychlost se může měnit vykonáním další funkce `IndAMove` (nebo další instrukce `IndXMove`). Když je zvolena rychlost v opačném směru, osa se zastaví a potom zrychlí k nové rychlosti a směru.

U krokového vykonávání instrukce se osa nastavuje jen do nezávislého režimu. Osa začíná svůj pohyb, když je vykonávána další instrukce, a pokračuje tak dlouho, dokud trvá vykonávání programu. Více informací najdete v *Referenční příručka RAPID - Přehled RAPID*, sekce *Pohyb a principy I/O - Polohování během vykonávání programu - Nezávislé osy*.

Když je ukazatel programu posunut na začátek programu nebo k nové rutině, všechny osy jsou automaticky nastaveny na normál bez změny měřicího systému (ekvivalent k vykonávání instrukce `IndReset\Old`).

POZNÁMKA

Výsledkem instrukce `IndAMove` po operaci `IndCMove` může být otáčení osy zpět k pohybu prováděnému v instrukci `IndCMove`. Aby se tomu zabránilo, použijte instrukci `IndReset` před `IndAMove` nebo použijte instrukci `IndRMove`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Osy v nezávislém režimu nemohou být posunovány ručně (jog). Jestliže je podniknut pokus vykonávat osu ručně, osa se nebude pohybovat a zobrazí se chybová zpráva. Proveďte instrukci `IndReset` nebo posuňte ukazatel programu na main, aby nezávislý režim byl opuštěn.

Jestliže nastane výpadek napájení, když je osa v nezávislém režimu, program není možné restartovat. Zobrazí se chybová zpráva a program je třeba spustit od začátku.

Instrukce není vhodná pro spřažené osy zápěstí robota (viz *Referenční příručka RAPID - Přehled RAPID*, sekce *Principy pohybu a I/O - Polohování během vykonávání programu - Nezávislé osy*).

Další příklady

Více příkladů instrukce `IndAMove` je názorně uvedeno dole.

Příklad 1

```
ActUnit Station_A;
weld_stationA;
IndAMove Station_A,1\ToAbsNum:=90,20\Ramp:=50;
ActUnit Station_B;
weld_stationB_1;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
DeactUnit Station_A;
weld_stationB_2;
```

`Station_A` je aktivována a svařování bylo spuštěno na stanici A.

`Station_A` (osa 1) je potom posunuta na pozici 90 stupňů, zatímco robot svařuje na stanici B. Rychlost osy je 20 stupňů/sek. Rychlost se mění snížením zrychlení/zpomalení na 50 % max výkonu.

Když stanice A dosáhne této pozice, je deaktivována a nové načítání může probíhat na stanici ve stejné době, kdy robot pokračuje se svařováním na stanici B.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_AXIS_ACT`	Osa není aktivována.

Syntaxe

```
IndAMove
  [ MecUnit':=' ] < variable (VAR) of mecunit>' ,'
  [ Axis':=' ] < expression (IN) of num>
  [ '\'ToAbsPos':=' < expression (IN) of robtarget> ]
  | [ '\' ToAbsNum':=' < expression (IN) of num> ] ','
  [ Speed ':=' ] < expression (IN) of num>
  [ '\' Ramp':=' < expression (IN) of num > ] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.104 IndAMove - Nezávislý pohyb absolutní pozice

Independent Axis

Pokračování

Související informace

Pro informace o	Viz
Nezávislé osy všeobecně	*Technická referenční příručka - Přehled RAPID*
Independent Axis	*Application manual - Controller software IRC5*
Změna zpět do normálního režimu	*IndReset - Nezávislý reset na str 263*
Resetovat měřicí systém	*IndReset - Nezávislý reset na str 263*
Pohyb ostatních nezávislých os	*IndRMove - Nezávislý pohyb ve vztahu k pozici na str 267* *IndDMove - Nezávislý pohyb delta pozice na str 260* *IndCMove - Nezávislý soustavný pohyb na str 257*
Zkontrolovat stav rychlosti u nezávislých os	*IndSpeed - Status nezávislé rychlosti na str 1202*
Zkontrolovat stav pozice u nezávislých os	*IndInpos - Nezávislá osa v pozičním statutu na str 1200*
Definování nezávislých spojů	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.105 IndCMove - Nezávislý soustavný pohyb

Použití

`IndCMove`(*Independent Continuous Movement*) se používá pro změnu osy do nezávislého režimu a spuštění soustavného pohybu osy určenou rychlostí.

Nezávislá osa je taková osa, která se pohybuje nezávisle na ostatních osách v robotickém systému. Jelikož vykonávání programu pokračuje okamžitě, je možné vykonat jiné instrukce (včetně polohovacích instrukcí) během času, kdy se nezávislá osa pohybuje.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `IndCMove`:

Viz také *Další příklady na str 258*.

Příklad 1

```
IndCMove Station_A,2,-30.5;
```

Osa `2` od `Station_A`se začíná pohybovat v záporném směru rychlostí 30,5 stupně/sek.

Argumenty

```
IndCMove MecUnit Axis Speed [\Ramp]
```

`MecUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`Axis`

Datový typ: num

Číslo aktuální osy pro mechanickou jednotku (1-6).

`Speed`

Datový typ: `num`

Rychlost osy ve stupních (mm/s u lineární osy).

Směr pohybu je stanoven se značkou argumentu rychlosti.

`[\Ramp]`

Datový typ: `num`

Snížit zrychlení a zpomalení od max výkonu (1-100%,100%=maximumperformance).

Vykonávání programu

Když je vykonáván `IndCMove`, určená osa se začíná pohybovat naprogramovanou rychlostí. Směr pohybu je určen jako znaménko pro argument rychlosti. Jestliže je naprogramován `\Ramp`, dojde ke snížení zrychlení/zpomalení.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.105 IndCMove - Nezávislý soustavný pohyb
Independent Axis
Pokračování

Pro změnu osy zpět do normálního režimu se použije instrukce `IndReset`. Ve spojení s tím může být změněna logická pozice osy tak, počet plných otáček může být vymazán, například, aby se zabránilo otáčení zpět k dalšímu pohybu.

Rychlost je možné změnit vykonáním další instrukce `IndCMove`. Jestliže je přikázána rychlost v opačném směru, osa se zastaví a potom zrychluje k nové rychlosti a směru. K zastavení osy může být použit argument rychlosti 0. Potom bude stále v nezávislém režimu.

Během krokového vykonávání instrukce se osa nastavuje jen do nezávislého režimu. Osa začíná svůj pohyb, když je vykonávána další instrukce, a pokračuje tak dlouho, dokud trvá vykonávání programu. Více informací najdete v *Referenční příručka RAPID - Přehled RAPID*, sekce *Pohyb a principy I/O - Polohování během vykonávání programu - Nezávislé osy*.

Když je ukazatel programu posunut na začátek programu nebo k nové rutině, všechny osy jsou automaticky nastaveny na normální režim bez změny měřicího systému (ekvivalent k vykonávání instrukce `IndReset\Old`).

Omezení

Rozlišení pozice osy se zhoršuje, čím je posunuta dál od své logické nulové pozice (obvykle střed pracovní oblasti). Abychom získali vysoké rozlišení zpět, můžeme nastavit logickou pracovní oblast na nulu s instrukcí `IndReset`. Více informací najdete zde: *Referenční příručka RAPID - Přehled RAPID*, sekce *Principy pohybu a I/O - Polohování během vykonávání programu - Nezávislé osy*.

Osy v nezávislém režimu nemohou být posunovány ručně (jog). Jestliže je podniknut pokus vykonávat osu ručně, osa se nebude pohybovat a zobrazí se chybová zpráva. Proveďte instrukci `IndReset` nebo posuňte ukazatel programu na main, aby nezávislý režim byl opuštěn.

Jestliže nastane výpadek napájení, když je osa v nezávislém režimu, program není možné restartovat. Zobrazí se chybová zpráva a program je třeba spustit od začátku.

Instrukce není vhodná pro spřažené osy zápěstí robota (viz *Referenční příručka RAPID - Přehled RAPID*, sekce *Principy pohybu a I/O - Polohování během vykonávání programu - Nezávislé osy*).

Další příklady

Více příkladů instrukce `IndCMove` je názorně uvedeno dole.

```
IndCMove Station_A,2,20;
WaitUntil IndSpeed(Station_A,2 \InSpeed) = TRUE;
WaitTime 0.2;
MoveL p10, v1000, fine, tool1;
IndCMove Station_A,2,-10\Ramp:=50;
MoveL p20, v1000, z50, tool1;
IndRMove Station_A,2 \ToRelPos:=p1 \Short,10;
MoveL p30, v1000, fine, tool1;
WaitUntil IndInpos(Station_A,2 ) = TRUE;
WaitTime 0.2;
IndReset Station_A,2 \RefPos:=p40\Short;
MoveL p40, v1000, fine, tool1;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Osa `2` od `Station_A` se začíná pohybovat s kladném směru rychlostí `20` stupňů/sek. Když tato osa dosáhne zvolené rychlosti, osy robotu se začnou pohybovat.

Když robot dosáhne pozice `p10`, externí osa změní směr a otáčí se rychlostí `10` stupňů/sek. Změna rychlosti se provádí sníženým zrychlením/zpomalením na `50`% maximální hodnoty. Současně robot provádí směrem k `p20`.

Osa `2` od `Station_A` je potom zastavena co nejrychleji v pozici `p1` v rámci aktuální otáčky.

Když osa 2 dosáhla své pozice a robot se zastavil v pozici `p30`, osa `2` se vrací znovu do normálního režimu. Ofset měřicího systému pro tuto osu se změní o celý počet otáček osy, takže aktuální pozice je co nejblíže k `p40`.

Když je robot potom posunut do pozice `p40`, osa `2` od `Station_A` bude posunuta instrukcí `MoveL p40` nejkratší trasou do pozice `p40` (max ±180 stupňů).

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_AXIS_ACT`	Osa není aktivována.

Syntaxe

```
IndCMove
  [ MecUnit':=' ] < variable (VAR) of mecunit>' ,'
  [ Axis':=' ] < expression (IN) of num> ',' 
  [ Speed ':=' ] < expression (IN) of num>
  [ '\' Ramp':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Nezávislé osy všeobecně	*Technická referenční příručka - Přehled RAPID*
Independent Axis	*Application manual - Controller software IRC5*
Změna zpět do normálního režimu	*IndReset - Nezávislý reset na str 263*
Resetovat měřicí systém	*IndReset - Nezávislý reset na str 263*
Pohyb ostatních nezávislých os	*IndAMove - Nezávislý pohyb absolutní pozice na str 253* *IndRMove - Nezávislý pohyb ve vztahu k pozici na str 267* *IndDMove - Nezávislý pohyb delta pozice na str 260*
Zkontrolovat stav rychlosti u nezávislých os	*IndSpeed - Status nezávislé rychlosti na str 1202*
Zkontrolovat stav pozice u nezávislých os	*IndInpos - Nezávislá osa v pozičním statutu na str 1200*
Definování nezávislých spojů	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.106 IndDMove - Nezávislý pohyb delta pozice

Použití

`IndDMove`(*Independent Delta Movement*) se používá pro změnu osy do nezávislého režimu a posunutí osy do určitého cíle.

Nezávislá osa je taková osa, která se pohybuje nezávisle na ostatních osách v robotickém systému. Jelikož vykonávání programu pokračuje okamžitě, je možné vykonat jiné instrukce (včetně polohovacích instrukcí) během času, kdy se nezávislá osa pohybuje.

Jestliže má být osa posunuta do určité pozice, musí být místo toho použita instrukce `IndAMove` nebo `IndRMove`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `IndDMove`:

Viz také *Další příklady na str 261*.

Příklad 1

```
IndDMove Station_A,2,-30,20;
```

Osa `2` od `Station_A` je posunuta o `30` stupňů v záporném směru rychlostí `20` stupňů/sek.

Argumenty

```
IndDMove MecUnit Axis Delta Speed [\Ramp]
```

`MecUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`Axis`

Datový typ: `num`

Číslo aktuální osy pro mechanickou jednotku (1-6).

`Delta`

Datový typ: `num`

Vzdálenost, na kterou by měla být posunuta aktuální osa, vyjádřená ve stupních (mm u lineárních os). Znaménko určuje směr pohybu.

`Speed`

Datový typ: `num`

Rychlost osy ve stupních (mm/s u lineární osy).

`[ \Ramp ]`

Datový typ: `num`

Snížit zrychlení a zpomalení od max výkonu (1-100%,100%=maximumperformance).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Když je vykonáván `IndDMove`, určená osa se pohybuje naprogramovanou rychlostí k určenému cíli. Směr pohybu je určen jako znaménko pro argument `Delta`. Jestliže je naprogramován `\Ramp`, dojde ke snížení zrychlení/zpomalení.

Jestliže se osa pohybuje, je vypočítána nová pozice z momentální pozice osy, když je vykonána instrukce `IndDMove`. Jestliže je vykonávána instrukce `IndDMove` se vzdáleností 0 a osa má již pohyblivou pozici, osa se zastaví a potom se posune zpět na pozici, kterou měla osa při vykonávání instrukce.

Pro změnu osy zpět do normálního režimu se použije instrukce `IndReset`. Ve spojení s tím může být změněna logická pozice osy tak, počet plných otáček může být z pozice vymazán, například, aby se zabránilo otáčení zpět k dalšímu pohybu.

Rychlost se může měnit vykonáním další funkce `IndDMove` (nebo další instrukce `IndXMove`). Když je zvolena rychlost v opačném směru, osa se zastaví a potom zrychlí k nové rychlosti a směru.

Během krokového vykonávání instrukce se osa nastavuje jen do nezávislého režimu. Osa začíná svůj pohyb, když je vykonávána další instrukce, a pokračuje tak dlouho, dokud trvá vykonávání programu. Více informací najdete v *Referenční příručka RAPID - Přehled RAPID*, sekce *Pohyb a principy I/O - Polohování během vykonávání programu - Nezávislé osy*.

Když je ukazatel programu posunut na začátek programu nebo k nové rutině, všechny osy jsou automaticky nastaveny na normální režim bez změny měřicího systému (ekvivalent k provádění instrukce `IndReset \Old`).

Omezení

Osy v nezávislém režimu nemohou být posunovány ručně (jog). Jestliže je podniknut pokus vykonávat osu ručně, osa se nebude pohybovat a zobrazí se chybová zpráva. Proveďte instrukci `IndReset` nebo posuňte ukazatel programu na main, aby nezávislý režim byl opuštěn.

Jestliže nastane výpadek napájení, když je osa v nezávislém režimu, program není možné restartovat. Zobrazí se chybová zpráva a program je třeba spustit od začátku.

Instrukce není vhodná pro spřažené osy zápěstí robota (viz *Referenční příručka RAPID - Přehled RAPID*, sekce *Principy pohybu a I/O - Polohování během vykonávání programu - Nezávislé osy*).

Další příklady

Více příkladů instrukce `IndDMove` je názorně uvedeno dole.

Příklad 1

```
IndAMove ROB_1,6\ToAbsNum:=90,20;
WaitUntil IndInpos(ROB_1,6) = TRUE;
WaitTime 0.2;
IndDMove Station_A,2,-30,20;
WaitUntil IndInpos(ROB_1,6) = TRUE;
WaitTime 0.2;
IndDMove ROB_1,6,400,20;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.106 IndDMove - Nezávislý pohyb delta pozice

Independent Axis
Pokračování

Osa 6 robotu je posunuta k následujícím pozicím:

- 90 stupňů
- 60 stupňů
- 460 stupňů (1 otáčka + 100 stupňů)

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_AXIS_ACT	Osa není aktivována.

Syntaxe

```
IndDMove
[ MecUnit':=' ] < variable (VAR) of mecunit> ','
[ Axis':=' ] < expression (IN) of num> ','
[ Delta':=' ] < expression (IN) of num>','
[ Speed ':=' ] < expression (IN) of num>
[ '\' Ramp':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Nezávislé osy všeobecně	*Technická referenční příručka - Přehled RAPID*
Independent Axis	*Application manual - Controller software IRC5*
Změna zpět do normálního režimu	*IndReset - Nezávislý reset na str 263*
Resetovat měřicí systém	*IndReset - Nezávislý reset na str 263*
Pohyb ostatních nezávislých os	*IndAMove - Nezávislý pohyb absolutní pozice na str 253* *IndRMove - Nezávislý pohyb ve vztahu k pozici na str 267* *IndCMove - Nezávislý soustavný pohyb na str 257*
Zkontrolovat stav rychlosti u nezávislých os	*IndSpeed - Status nezávislé rychlosti na str 1202*
Zkontrolovat stav pozice u nezávislých os	*IndInpos - Nezávislá osa v pozičním statutu na str 1200*
Definování nezávislých spojů	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.107 IndReset - Nezávislý reset

Použití

`IndReset` (*Independent Reset*) se používá pro změnu nezávislé osy zpět do normálního režimu. Současně může být měřicí systém pro otočné osy posunut o jistý počet otáček osy.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `IndReset`:

Viz také *Další příklady na str 265*.

```
IndCMove Station_A,2,5;
MoveL *,v1000,fine,tool1;
IndCMove Station_A,2,0;
WaitUntil IndSpeed(Station_A,2\ZeroSpeed);
WaitTime 0.2
IndReset Station_A,2;
```

Osa `2` od `Station_A` je nejprve posunuta v nezávislém režimu a potom změněna zpět do normálního režimu. Osa si udrží svoji pozici.

> **POZNÁMKA**
>
> Aktuální nezávislá osa a normální osy by se neměly pohybovat, když je vykonávána instrukce `IndReset`. Proto je předchozí pozice stop bodem a instrukce `IndCMove` je vykonávána nulovou rychlostí. Dále, pauza 0,2 sek. se používá k ujištění, že bylo dosaženo správného stavu.

Argumenty

```
IndReset MecUnit Axis [\RefPos] | [\RefNum] [\Short] | [\Fwd]
     |[\Bwd] | \Old]
```

`MecUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`Axis`

Datový typ: `num`

Číslo aktuální osy pro mechanickou jednotku (1-6).

`[ \RefPos ]`

Reference Position

Datový typ: `robtarget`

Referenční pozice osy určená jako `robtarget`. Používá se pouze komponent pro tuto konkrétní `Axis`. Pozice musí být uvnitř normálního pracovního rozsahu.

Pro osy robotu se místo toho použije argument `\RefNum`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.107 IndReset - Nezávislý reset
Independent Axis
Pokračování

Argument se definuje pouze společně s argumentem `\Short`, `\Fwd` nebo `\Bwd`. Není přípustný společně s argumentem `\Old`.

`[ \RefNum ]`

Reference Numeric value

Datový typ: `num`

Referenční pozice osy definovaná ve stupních (mm u lineárních os). Pozice musí být uvnitř normálního pracovního rozsahu.

Argument se definuje pouze společně s argumentem `\Short`, `\Fwd` nebo `\Bwd`. Není přípustný společně s argumentem `\Old`.

Stejná funkce jako `\RefPos`, ale pozice je definována jako numerická hodnota kvůli usnadnění ruční změny pozice.

`[ \Short ]`

Datový typ: `switch`

Měřicí systém změní plný počet otáček na straně osy, takže osa bude co nejblíže k určené pozici `\RefPos` nebo `\RefNum`. Jestliže polohovací instrukce se stejnou pozicí je vykonávána po `IndReset`, osa bude postupovat nejkratší trasou, méně než ±180 stupňů, aby této pozice dosáhla.

`[ \Fwd ]`

Forward

Datový typ: `switch`

Měřicí systém změní plný počet otáček na straně osy, takže referenční pozice bude na kladné straně určené pozice `\RefPos` nebo `\RefNum`. Jestliže polohovací instrukce se stejnou pozicí je vykonávána po `IndReset`, osa se otočí v kladném směru méně než 360 stupňů, aby této pozice dosáhla.

`[ \Bwd ]`

Backward

Datový typ: `switch`

Měřicí systém změní plný počet otáček na straně osy, takže referenční pozice bude na záporné straně určené pozice `\RefPos` nebo `\RefNum`. Jestliže polohovací instrukce se stejnou pozicí je vykonávána po `IndReset`, osa se otočí v záporném směru méně než 360 stupňů, aby této pozice dosáhla.

`[ \Old ]`

Datový typ: `switch`

Udržuje si starou pozici.

POZNÁMKA

Rozlišení se snižuje v pozicích daleko od nuly.

Jestli není určen žádný argument `\Short`, `\Fwd`, `\Bwd` nebo `\Old`, jako výchozí hodnota se používá `\Old`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Když je proveden IndResetIndReset (`IndReset`), mění nezávislou osu zpět do normálního režimu. Současně může být měřicí systém pro osu posunut o plný počet otáček osy.

Instrukci je možné používat také v normálním režimu ke změně měřicího systému.

> **POZNÁMKA**
>
> Pozice se používá pouze pro úpravu měřicího systému - osa se neposune k pozici.

Omezení

Instrukci je možné vykonávat pouze když všechny aktivní osy běžící v normálním režimu jsou v klidu. Všechny aktivní osy v každé mechanické jednotce připojené ke stejnému plánovači pohybu musí stát v klidu. Osa v nezávislém režimu, která bude změněna do normálního režimu, musí být také stacionární. U os v normálním režimu se toho dosáhne vykonáním pohybové instrukce s argumentem `fine`. Nezávislou osu zastaví instrukce `IndCMove` s `Speed:=0` (následováno dobou čekání 0,2 sek.), `IndRMove`, `IndAMove`, nebo `IndDMove`.

Rozlišení pozic se snižuje při vzdalování od logické pozice 0. Osa, která se postupně otáčí dál a dál od pozice 0, by tedy měla být nastavena na nulu pomocí instrukce `IndReset` s argumentem jiným než `\Old`.

Měřicí systém není možné měnit u lineárních os.

Pro zajištění řádného startu po `IndReset` osy s relativně změřeným měřicím systémem (spínače synchronizace) musí být přidána zvláštní časová prodleva 0,12 sek. po instrukci `IndReset`.

Pouze osa robotu 6 se může použít jako nezávislá osa. Instrukce `IndReset` se může použít také pro osu 4 na modelech IRB 1600, 2600 a 4600 (nikoliv verze ID). Jestliže se použije `IndReset` pro osu 4 robotu, potom osa 6 nesmí být v nezávislém režimu.

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (`zonedata fine`), nikoliv s fly-by bodem, jinak nebude možný restart po selhání napájení.

`IndReset` se nemůže provádět v RAPID rutině připojené ke kterékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart nebo Step.

`IndReset` pouze přepíná nezávislý stav pro osu. Nemůže se používat pro zastavení nezávislého pohybu. Aby bylo možné nezávislý pohyb zastavit, je třeba dosáhnout stop podmínky nebo uživatel musí, například, posunout PP na main.

Další příklady

Více příkladů instrukce `IndReset` je názorně uvedeno dole.

Příklad 1

```
IndAMove Station_A,1\ToAbsNum:=750,50;
WaitUntil IndInpos(Station_A,1);
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.107 IndReset - Nezávislý reset

Independent Axis

Pokračování

```
WaitTime 0.2;
IndReset Station_A,1 \RefNum:=0 \Short;
IndAMove Station_A,1\ToAbsNum:=750,50;
WaitUntil IndInpos(Station_A,1);
WaitTime 0.2;
IndReset Station_A,1 \RefNum:=300 \Short;
```

Osa `1` v `Station_A` se nejdříve přesune nezávisle na pozici `750` stupňů (2 otáčky a 30 stupňů). Současně, jak proběhne změna na normální režim, je logická pozice nastavena na 30 stupňů.

Osa `1` v `Station_A` je následně přesunuta na pozici `750` stupňů (2 otáčky a 30 stupňů). Současně, jak proběhne změna na normální režim, je logická pozice nastavena na 390 stupňů (1 otáčka a 30 stupňů).

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_AXIS_ACT`	Osa není aktivována.
`ERR_AXIS_MOVING`	Osa se pohybuje.

Syntaxe

```
IndReset
  [ MecUnit':=' ] < variable (VAR) of mecunit> ','
  [ Axis ':=' ] < expression (IN) of num>
  [ '\' RefPos ':=' < expression (IN) of robtarget> ] |
  [ '\' RefNum ':=' < expression (IN) of num> ]
  [ '\' Short ] | [ '\' Fwd ] | [ '\' Bwd ] | [ '\' Old ]';'
```

Související informace

Pro informace o	Viz
Nezávislé osy všeobecně	*Technická referenční příručka - Přehled RAPID*
Independent Axis	*Application manual - Controller software IRC5*
Změnit osu na nezávislý režim	*IndAMove - Nezávislý pohyb absolutní pozice na str 253* *IndCMove - Nezávislý soustavný pohyb na str 257* *IndDMove - Nezávislý pohyb delta pozice na str 260* *IndRMove - Nezávislý pohyb ve vztahu k pozici na str 267*
Zkontrolovat stav rychlosti u nezávislých os	*IndSpeed - Status nezávislé rychlosti na str 1202*
Zkontrolovat stav pozice u nezávislých os	*IndInpos - Nezávislá osa v pozičním statutu na str 1200*
Definování nezávislých spojů	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.108 IndRMove - Nezávislý pohyb ve vztahu k pozici

Použití

`IndRMove`(*Independent Relative Movement*) se používá pro změnu rotační osy do nezávislého režimu a posunutí osy na určitou pozici v rámci jedné otáčky.

Nezávislá osa je taková osa, která se pohybuje nezávisle na ostatních osách v robotickém systému. Jelikož vykonávání programu pokračuje okamžitě, je možné vykonat jiné instrukce (včetně polohovacích instrukcí) během času, kdy se nezávislá osa pohybuje.

Jestliže osa bude posunuta na absolutní pozici (několik otáček) nebo jestliže osa je lineární, měla by se místo toho použít instrukce `IndAMove`. Jestliže pohyb bude proveden na určitou vzdálenost od aktuální pozice, musí se použít instrukce `IndDMove`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `IndRMove`:

Viz také *Další příklady na str 269*.

Příklad 1

```
IndRMove Station_A,2\ToRelPos:=p5 \Short,20;
```

Osa `2` od `Station_A` je posunuta nejkratší trasou na pozici `p5` v rámci jedné otáčky (max otáčení +/- 180 stupňů) rychlostí `20` stupňů/sek.

Argumenty

```
IndRMove MecUnit Axis [\ToRelPos] | [\ToRelNum] [\Short] | [\Fwd]
     | [\Bwd] Speed [\Ramp]
```

`MecUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`Axis`

Datový typ: `num`

Číslo aktuální osy pro mechanickou jednotku (1-6).

`[ \ToRelPos ]`

To Relative Position

Datový typ: `robtarget`

Pozice osy určená jako `robtarget`. Používá se pouze komponent pro tuto konkrétní `Axis`. Hodnota se používá jako poziční hodnota ve stupních v rámci jedné otáčky osy. To znamená, že osa se pohybuje méně než jednu otáčku.

Pozice osy bude ovlivněna, jestliže osa je posunuta pomocí instrukce `EOffsSet` or `EOffsOn`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.108 IndRMove - Nezávislý pohyb ve vztahu k pozici

Independent Axis

Pokračování

Pro osy robotu se místo toho použije argument `\ToRelNum`.

`[ \ToRelNum ]`

To Relative Numeric value

Datový typ: `num`

Pozice osy definovaná ve stupních.

Pomocí tohoto argumentu NEBUDE pozice ovlivněna žádným posuvem, například `EOffsSet` nebo `PDispOn`.

Stejná funkce jako `\ToRelPos`, ale pozice je definována jako numerická hodnota kvůli usnadnění ruční změny pozice.

`[ \Short ]`

Datový typ: `switch`

Osa je posunuta nejkratší trasou na novou pozici. To znamená, že max rotace bude 180 stupňů v jakémkoliv směru. Směr pohybu proto závisí na aktuálním umístění osy.

`[ \Fwd ]`

Forward

Datový typ: `switch`

Osa je přesunuta v kladném směru na novou pozici. To znamená, že max rotace bude 360 stupňů a vždy v kladném směru (zvýšená hodnota pozice).

`[ \Bwd ]`

Backward

Datový typ: `switch`

Osa je přesunuta v záporném směru na novou pozici. To znamená, že max rotace bude 360 stupňů a vždy v záporném směru (snížená hodnota pozice).

Jestliže argument `\Short`, `\Fwd` nebo `\Bwd` byl vypuštěn, jako výchozí hodnota se používá `\Short`.

`Speed`

Datový typ: `num`

Rychlost osy ve stupních/sek.

`[ \Ramp ]`

Datový typ: `num`

Snížit zrychlení a zpomalení od max výkonu (1-100%,100%=maximumperformance).

Vykonávání programu

Když je vykonáván `IndRMove`, určená osa se pohybuje naprogramovanou rychlostí k určené pozici osy, ale pouze max jednu otáčku. Jestliže je naprogramován `\Ramp`, dojde ke snížení zrychlení/zpomalení.

Pro změnu osy zpět do normálního režimu se použije instrukce `IndReset`. Ve spojení s tím může být změněna logická pozice osy tak, počet plných otáček může být z pozice vymazán, například, aby se zabránilo otáčení zpět k dalšímu pohybu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Rychlost se může měnit vykonáním další funkce `IndRMove` (nebo další instrukce `IndXMove`). Když je zvolena rychlost v opačném směru, osa se zastaví a potom zrychlí k nové rychlosti a směru.

Během krokového vykonávání instrukce se osa nastavuje jen do nezávislého režimu. Osa začíná svůj pohyb, když je vykonávána další instrukce, a pokračuje tak dlouho, dokud trvá vykonávání programu. Více informací najdete v *Referenční příručka RAPID - Přehled RAPID*, sekce *Principy pohybu a I/O - Polohování během vykonávání programu - Nezávislé osy*.

Když je ukazatel programu posunut na začátek programu nebo k nové rutině, všechny osy jsou automaticky nastaveny na normální režim bez změny měřicího systému (ekvivalent k provádění instrukce `IndReset \Old`).

Omezení

Osy v nezávislém režimu nemohou být posunovány ručně (jog). Jestliže je podniknut pokus vykonávat osu ručně, osa se nebude pohybovat a zobrazí se chybová zpráva. Proveďte instrukci `IndReset` nebo posuňte ukazatel programu na main, aby nezávislý režim byl opuštěn.

Jestliže nastane výpadek napájení, když je osa v nezávislém režimu, program není možné restartovat. Zobrazí se chybová zpráva a program je třeba spustit od začátku.

Instrukce není vhodná pro spřažené osy zápěstí robota (viz *Referenční příručka RAPID - Přehled RAPID*, sekce *Principy pohybu a I/O - Polohování během vykonávání programu - Nezávislé osy*).

Další příklady

Více příkladů instrukce `IndRMove` je názorně uvedeno dole.

Příklad 1

```
IndRMove Station_A,1\ToRelPos:=p5 \Fwd,20\Ramp:=50;
```

Osa `1` od `Station_A` se začíná pohybovat v kladném směru k pozici `p5` v rámci jedné otáčky (max otáčky 360 stupňů) rychlostí `20` stupňů/sek. Rychlost se mění se zrychlením/zpomalením sníženým na `50`% maximálního výkonu.

```
IndAMove Station_A,1\ToAbsNum:=90,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
IndRMove Station_A,1\ToRelNum:=80 \Fwd,20;
WaitTime 0.2;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
IndRMove Station_A,1\ToRelNum:=50 \Bwd,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
IndRMove Station_A,1\ToRelNum:=150 \Short,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
IndAMove Station_A,1\ToAbsNum:=10,20;
```

Osa `1` od `Station_A` je posunuta k následujícím pozicím:

- 90 stupňů
- 440 stupňů (1 otáčka + 80 stupňů)

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.108 IndRMove - Nezávislý pohyb ve vztahu k pozici

Independent Axis

Pokračování

- 410 stupňů (1 otáčka + 50 stupňů)
- 510 stupňů (1 otáčka + 150 stupňů)
- 10 stupňů

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_AXIS_ACT	Osa není aktivována.

Syntaxe

```
IndRMove
  [ MecUnit ':=' ] < variable (VAR) of mecunit> ','
  [ Axis ':=' ] < expression (IN) of num>
  [ '\' ToRelPos ':=' < expression (IN) of robtargets> ]
  | [ '\' ToRelNum ':=' < expression (IN) of num> ]
  [ '\' Short ] | [ '\' Fwd ] | [ '\' Bwd ] ','
  [ Speed ':=' ] < expression (IN) of num>
  [ '\' Ramp ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Nezávislé osy všeobecně	*Technická referenční příručka - Přehled RAPID*
Independent Axis	*Application manual - Controller software IRC5*
Změna zpět do normálního režimu	*IndReset - Nezávislý reset na str 263*
Resetovat měřicí systém	*IndReset - Nezávislý reset na str 263*
Pohyb ostatních nezávislých os	*IndAMove - Nezávislý pohyb absolutní pozice na str 253* *IndDMove - Nezávislý pohyb delta pozice na str 260* *IndCMove - Nezávislý soustavný pohyb na str 257*
Zkontrolovat stav rychlosti u nezávislých os	*IndSpeed - Status nezávislé rychlosti na str 1202*
Zkontrolovat stav pozice u nezávislých os	*IndInpos - Nezávislá osa v pozičním statutu na str 1200*
Definování nezávislých spojů	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.109 InitSuperv - Resetovat veškerý dohled pro CAP

Použití

`InitSuperv` se používá pro iniciaci CAP dohledu. To znamená, že všechny seznamy dohledu budou vyčištěny a všechny záznamy I/O budou odstraněny.

Příklad

```
PROC main()
  InitSuperv;
  SetupSuperv diWR_EST, ACT,SUPERV_MAIN;
  SetupSuperv diGA_EST, ACT,SUPERV_MAIN;
  CapL p2, v100, cdata1, weavestart, weave,fine, tWeldGun;
ENDPROC
```

`InitSuperv` se používá pro vyčištění všech seznamů dohledu před nastavením nového dohledu.

Omezení

Instrukce `InitSuperv` by měla být vykonána pouze jednou, například ze zakládací přihrádky.

Syntaxe

```
InitSuperv ';'
```

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instrukce `SetupSuperv`	*SetupSuperv - Nastavit podmínky pro dohled signálu v CAP na str 639*
Instrukce `RemoveSuperv`	*RemoveSuperv - Odstranit podmínku pro jeden signál na str 538*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.110 InvertDO - Invertuje hodnotu digitálního výstupního signálu

Použití

`InvertDO` (*Invert Digital Output*) invertuje hodnotu digitálního výstupního signálu (0 -> 1 a 1 -> 0).

Základní příklady

Následující příklad názorně ukazuje instrukci `InvertDO`:

Příklad 1

```
InvertDO do15;
```

Aktuální hodnota signálu `do15` je invertována .

Argumenty

```
InvertDO Signal
```

`Signal`

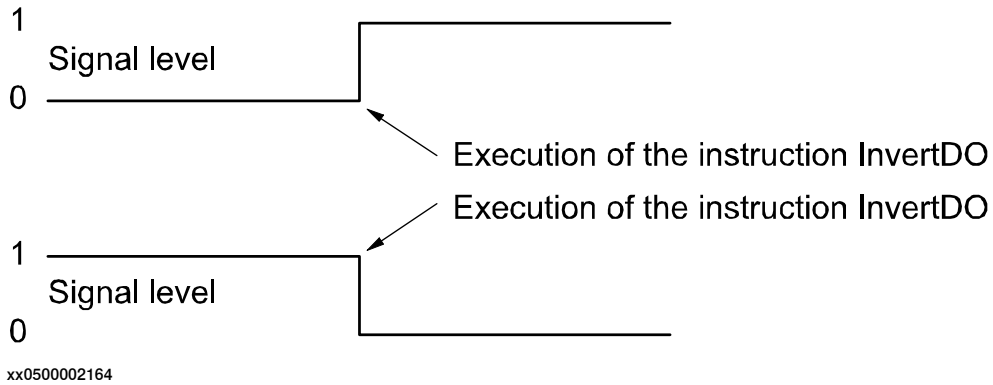
Datový typ: `signaldo`

Jméno signálu, který bude invertován.

Vykonávání programu

Aktuální hodnota signálu je invertována (viz obrázek dole).

Obrázek dole ukazuje inverzi digitálního výstupního signálu.



xx0500002164

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_NO_ALIASIO_DEF`	Proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.
`ERR_NORUNUNIT`	Není žádný kontakt s jednotkou I/O.
`ERR_SIG_NOT_VALID`	Není přístup k I/O signálu (platí pouze pro aplikační sběrnici ICI).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
InvertDO
  [ Signal ':=' ] < variable (VAR) of signaldo > ';'
```

Související informace

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.111 IOBusStart - Start of I/O bus

Použití

`IOBusStart` se používá pro spuštění určité I/O sběrnice.

Základní příklady

Následující příklad názorně ukazuje instrukci `IOBusStart`:

Příklad 1

```
IOBusStart "IBS";
```

Instrukce spouští I/O sběrnici s jménem `IBS`.

Argumenty

```
IOBusStart BusName
```

`BusName`

Datový typ: `string`

Jméno I/O sběrnice která bude spuštěna.

Vykonávání programu

Spustit I/O sběrnici se jménem stanoveným v parametru `BusName`.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_NAME_INVALID`	Jméno I/O sběrnice neexistuje.

Syntaxe

```
IOBusStart
  [ BusName ':=' ] < expression (IN) of string>';'
```

Související informace

Pro informace o	Viz
Jak získat stav I/O sběrnice	*IOBusState - Získat aktuální stav I/O sběrnice na str 275*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.112 IOBusState - Získat aktuální stav I/O sběrnice

Použití

`IOBusState` se používá k přečtení stavu určité I/O sběrnice. Její fyzický stav a logický stav definují status pro I/O sběrnici.

Základní příklady

Následující příklady názorně ukazují instrukci `IOBusState`:

Příklad 1

```
VAR busstate bstate;

IOBusState "IBS", bstate \Phys;
TEST bstate
CASE IOBUS_PHYS_STATE_RUNNING:
  ! Possible to access the signals on the IBS bus
DEFAULT:
  ! Actions for not up and running IBS bus
ENDTEST
```

Instrukce vrací fyzický stav I/O sběrnice `IBS` v proměnné `bstate` typu `busstate`.

Příklad 2

```
VAR busstate bstate;

IOBusState "IBS", bstate \Logic;
TEST bstate
CASE IOBUS_LOG_STATE_STARTED:
  ! The IBS bus is started
DEFAULT:
  ! Actions for stopped IBS bus
ENDTEST
```

Instrukce vrací logický stav I/O sběrnice `IBS` v proměnné `bstate` typu `busstate`.

Argumenty

```
IOBusState BusName State [\Phys] | [\Logic]
```

`BusName`

Datový typ: `string`

Jméno I/O sběrnice, od které je třeba získat stav.

`State`

Datový typ: `busstate`

Proměnná, ve které je vrácen stav I/O sběrnice. Viz předdefinovaná data typu `busstate` pod Vykonáváním programu.

`[\Phys]`

Physical

Datový typ: `switch`

Jestliže se používá tento parametr, je přečten fyzický stav I/O sběrnice.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.112 IOBusState - Získat aktuální stav I/O sběrnice
RobotWare - OS
Pokračování

`[\Logic]`

Logical

Datový typ: `switch`

Jestliže se používá tento parametr, je přečten logický stav I/O sběrnice.

Vykonávání programu

Vracení v parametru `State`, stav I/O sběrnice, které je určen v parametru `BusName`.

Logické stavy I/O sběrnice popisují stav a uživatel může přikázat sběrnici. Stav I/O sběrnice je definován v tabulce dole, při používání volitelného argumentu `\Logic`.

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
10	IOBUS_LOG_STATE_STOPPED	Sběrnice byla zastavena kvůli chybě 2)
11	IOBUS_LOG_STATE_STARTED	Sběrnice byla spuštěna 1)

Fyzický stav I/O sběrnice popisuje stav, že ovladač sběrnice může přikázat sběrnici. Stav I/O sběrnice je definován v tabulce dole, při používání volitelného argumentu `\Phys`.

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
20	IOBUS_PHYS_STATE_HALTED	Sběrnice byla zastavena 3)
21	IOBUS_PHYS_STATE_RUNNING	Sběrnice je spuštěna a běží 1)
22	IOBUS_PHYS_STATE_ERROR	Sběrnice napracuje 2)
23	IOBUS_PHYS_STATE_STARTUP	Sběrnice je v režimu startu, nekomunikuje s žádnou I/O jednotkou.
24	IOBUS_PHYS_STATE_INIT	Sběrnice je jen vytvořena 3)

POZNÁMKA

Stav I/O sběrnice je definován v tabulce dole, když se nepoužívá žádný z volitelných argumentů `\Phys` nebo `\Logic`.

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
0	BUSSTATE_HALTED	Sběrnice byla zastavena 3)
1	BUSSTATE_RUN	Sběrnice je spuštěna a běží 1)
2	BUSSTATE_ERROR	Sběrnice napracuje 2)
3	BUSSTATE_STARTUP	Sběrnice je v režimu startu, nekomunikuje s žádnou I/O jednotkou.
4	BUSSTATE_INIT	Sběrnice je pouze vytvořena 3)

1) Jestliže I/O sběrnice je zapnuta a běží, stav vrácený v argumentu `State` v instrukci `IOBusState` může být buď `IOBUS_LOG_STATE_STARTED`,

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

IOBUS_PHYS_STATE_RUNNING, nebo BUSSTATE_RUN, podle toho, jestli jsou volitelné parametry v IOBusState použity nebo nikoliv.

2) Jestliže I/O sběrnice je zastavena kvůli nějaké chybě, stav vrácený v argumentu State může být buď IOBUS_LOG_STATE_STOPPED, IOBUS_PHYS_STATE_ERROR, nebo BUSSTATE_ERROR, podle toho, jestli jsou volitelné parametry v IOBusState použity nebo nikoliv.

3) Není možné získat tento stav v programu RAPID s aktuální verzí Robotware - OS.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_NAME_INVALID	Jméno I/O sběrnice neexistuje.

Syntaxe

```
IOBusState
  [ BusName ':=' ] < expression (IN) of string> ','
  [ State ':=' ] < variable (VAR) of busstate>
  [ '\' Phys] | [ '\' Logic] ';'
```

Související informace

Pro informace o	Viz
Definice stavu I/O sběrnice	*busstate - Stav I/O sběrnice na str 1443*
Start I/O sběrnice	*IOBusStart - Start of I/O bus na str 274*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.113 IODisable - Deaktivovat I/O jednotku

Použití

`IODisable` se používá pro deaktivaci I/O jednotky během vykonávání programu.

I/O jednotky se automaticky aktivují po startu, jestliže jsou definovány v systémových parametrech. Jestliže je to z nějakého důvodu žádoucí, I/O jednotky se mohou deaktivovat nebo aktivovat během vykonávání programu.

POZNÁMKA

Není možné deaktivovat jednotku I/O s Unit Trustlevel nastaveným na Required.

Základní příklady

Následující příklad názorně ukazuje instrukci `IODisable`:

Viz také *Další příklady na str 279*.

Příklad 1

```
CONST string board1:="board1";
IODisable board1, 5;
```

Deaktivovat IO jednotku s jménem `board1`. Čekat max `5` sekund.

Argumenty

```
IODisable UnitName MaxTime
```

`UnitName`

Datový typ: `string`

Jméno I/O jednotky (jméno jednotky musí být přítomno v systémových parametrech).

`MaxTime`

Datový typ: `num`

Max přípustná doba čekání vyjádřená v sekundách. Jestliže tento čas vyprší předtím, než I/O jednotka dokončí deaktivační kroky, bude zavolán chybový handler, pokud je přítomen, s chybovým kódem `ERR_IODISABLE`.. Jestliže nemáme chybový handler, vykonávání programu se zastaví. Deaktivační kroky I/O jednotky budou vždy pokračovat bez ohledu na `MaxTime` nebo chybu.

Deaktivace I/O jednotky bude trvat asi 0-5 sek.

Vykonávání programu

Určená I/O jednotka začíná deaktivační kroky. Instrukce je připravena, když jsou deaktivační kroky dokončeny. Jestliže vyprší `MaxTime` předtím, než I/O jednotka dokončí deaktivační kroky, bude generována odstranitelná chyba.

Po deaktivaci I/O jednotky bude výsledkem jakéhokoliv nastavování výstupů této jednotky chyba.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_IODISABLE	Doba čekání vyprší dříve, než je I/O jednotka deaktivována.
ERR_NAME_INVALID	Jméno I/O jednotky neexistuje.
ERR_TRUSTLEVEL	I/O jednotka nemůže být aktivována, jestliže Unit Trustlevel je nastavena na Required.

Další příklady

Více příkladů instrukce IODisable je názorně uvedeno dole.

Příklad 1

```
PROC go_home()
  VAR num recover_flag :=0;
  ...
  ! Start to deactivate I/O unit board1
  recover_flag := 1;
  IODisable "board1", 0;
  ! Move to home position
  MoveJ home, v1000,fine,tool1;
  ! Wait until deactivation of I/O unit board1 is ready
  recover_flag := 2;
  IODisable "board1", 5;
  ...
  ERROR
    IF ERRNO = ERR_IODISABLE THEN
      IF recover_flag = 1 THEN
        TRYNEXT;
      ELSEIF recover_flag = 2 THEN
        IF RemaningRetries() > 0 THEN
          RETRY;
        ELSE
          RAISE;
        ENDIF
      ENDIF
    ELSE
      ErrWrite "IODisable error", "Not possible to deactivate I/O
           unit board1";
      Stop;
    ENDIF
ENDPROC
```

Kvůli ušetření doby cyklu je I/O jednotka board1 deaktivována během pohybu robotu k pozici home. S robotem na pozici home je proveden test, aby bylo zjištěno, jestli I/O jednotka board1 je a nebo není plně deaktivována. Po max počtu pokusů (4 s čekací dobou 5 sek.) se provádění robotu zastaví s chybovou zprávou.

Stejný princip se může použít s IOEnable (tím se ušetří více času cyklu ve srovnání s IODisable).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.113 IODisable - Deaktivovat I/O jednotku

RobotWare - OS

Pokračování

Syntaxe

```
IODisable
  [ UnitName ':=' ] < expression (IN) of string> ','
  [ MaxTime ':=' ] < expression (IN) of num> ';'
```

Související informace

Pro informace o	Viz
Aktivace jednotky I/O	*IOEnable - Aktivovat I/O jednotku na str 281*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkce Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.114 IOEnable - Aktivovat I/O jednotku

Použití

`IOEnable` se používá pro aktivaci I/O jednotky během vykonávání programu.

I/O jednotky se automaticky aktivují po startu, jestliže jsou definovány v systémových parametrech. Jestliže je to z nějakého důvodu žádoucí, I/O jednotky se mohou deaktivovat nebo aktivovat během vykonávání programu.

Činnost ovladače (řadiče) při aktivaci I/O jednotky závisí na definované Unit Trustlevel v systémových parametrech.

Základní příklady

Následující příklad názorně ukazuje instrukci `IOEnable`:

Viz také *Další příklady na str 282*.

Příklad 1

```
CONST string board1:="board1";
IOEnable board1, 5;
```

Aktivovat IO jednotku s jménem `board1`. Čekat max `5` sekund.

Argumenty

```
IOEnable UnitName MaxTime
```

`UnitName`

Datový typ: `string`

Jméno I/O jednotky (jméno I/O jednotky musí být přítomno v systémových parametrech).

`MaxTime`

Datový typ: `num`

Max přípustná doba čekání vyjádřená v sekundách. Jestliže tento čas vyprší předtím, než I/O jednotka dokončí aktivační kroky, bude zavolán chybový handler, pokud je přítomen, s chybovým kódem `ERR_IOENABLE`.. Jestliže nemáme chybový handler, vykonávání programu se zastaví. Aktivační kroky I/O jednotky budou vždy pokračovat bez ohledu na `MaxTime` nebo chybu.

Aktivace I/O jednotky bude trvat asi 2-5 sek.

Vykonávání programu

Určená I/O jednotka začíná aktivační kroky. Instrukce je připravena, když jsou aktivační kroky dokončeny. Jestliže vyprší `MaxTime` předtím, než I/O jednotka dokončí aktivační kroky, bude generována odstranitelná chyba.

Po sekvenci `IODisable` - `IOEnable` budou všechny výstupy na aktuální I/O jednotce nastaveny na staré hodnoty (před `IODisable`).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.114 IOEnable - Aktivovat I/O jednotku

RobotWare - OS

Pokračování

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_IOENABLE`	Doba čekání vyprší dříve, než je I/O jednotka aktivována.
`ERR_NAME_INVALID`	Jméno I/O jednotky neexistuje
`ERR_BUSSTATE`	I/O sběrnice je v chybovém stavu nebo vchází do chybového stavu před aktivací I/O jednotky.

Další příklady

`IOEnable` se může také používat pro kontrolu, jestli je některá I/O jednotka z nějakého důvodu odpojena.

Více příkladů jak používat instrukci `IOEnable` je názorně uvedeno dole.

Příklad 1

```
VAR num max_retry:=0;
...
IOEnable "board1", 0;
SetDO board1_sig3, 1;
...
ERROR
  IF ERRNO = ERR_IOENABLE THEN
    IF RemaningRetries() > 0 THEN
      WaitTime 1;
      RETRY;
    ELSE
      RAISE;
    ENDIF
  ELSE
    ErrWrite "IOEnable error", "Not possible to activate I/O unit
         board1";
    Stop;
  ENDIF
```

Před používáním signálů na I/O jednotce `board1` je proveden test vyzkoušením aktivace I/O jednotky s časovačem po `0` sek. Jestliže test selže, je proveden skok k chybovému handleru. V chybovém handleru čeká vykonávání programu `1` sekundu a je proveden nový pokus. Po `4` pokusech je chyba `ERR_IOENABLE` rozšířena k volajícímu této rutiny.

Syntaxe

```
IOEnable
  [ UnitName ':=' ] < expression (IN) of string>' ,'
  [ MaxTime' :=' ] < expression (IN) of num > ';'
```

Související informace

Pro informace o	Viz
Deaktivace jednotky I/O	*IODisable - Deaktivovat I/O jednotku na str 278*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.115 IPathPos - Získat robtarget střední linie, když probíhá weaving

Použití

`IPathPos` se používá pro vyhledání pozice střední linie během weavingu s CAP.

Tato funkce se používá většinou společně se sledovací funkčností (tracking). Je nezbytné aktivovat weaving a synchronizační signály na levé i pravé straně.

Základní příklad

```
connect intpt, TRP_ipathpos IPathPos p_robt, sen_pos, intpt;
```

Když `p_robt` dostává novou vypočítanou hodnotu, přerušení `intpt` bude odesláno a TRAP `TRP_ipathpos` bude vykonána.

Argumenty

```
IPathPos p_robt, sen_pos, intpt [\NoDispl]
```

p_robt

Datový typ: `robtarget`

`p_robt` udržuje poslední hodnotu vypočítaného `robtarget`.

sen_pos

Datový typ: `pos`

`sen_pos` se nepoužívá.

intpt

Datový typ: `intno`

`intpt` určuje přerušení, které bude přijato pokaždé, když je nová hodnota přidělena pro `p_robt`.

[\NoDispl]

Datový typ: `switch`

Jestliže je určen `\NoDispl`, vrácená hodnota v `PERS p_robt` nebude zahrnovat žádný posun, který může být určen pomocí instrukcí RAPID `PDispSet` a `PDispOn`.

Omezení

Je nezbytné aktivovat weaving a weave synchronizaci (s nebo bez sledování) (tracking).

Syntaxe

```
IPathPos
  [p_robt ':='] < persistent (PERS) of robtarget > ','
  [sen_pos ':='] < persistent (PERS) of pos > ','
  [Interrupt ':='] < variable (IN) of intnum >
  ['\' NoDispl ] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instrukce `CapWeaveSync`	*CapWeaveSync - signály nastavení a úrovní pro weave synchronizaci na str 100*
Instrukce `CapAPTrSetup`	*CapAPTrSetup - Nastavení At-Point-Tracker na str 66*
Instrukce `CapLATrSetup`	*CapLATrSetup - Nastavit Look-Ahead-Tracker na str 91*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.116 IPers - Přerušení při změně hodnoty proměnné perzistentu

Použití

`IPers` (*Interrupt Persistent*) se používá k přikázání a zapnutí přerušení, která budou generována, když je změněna hodnota proměnné perzistentu.

Základní příklady

Následující příklad názorně ukazuje instrukci `IPers`:

Příklad 1

```
VAR intnum pers1int;
PERS num counter := 0;

PROC main()
  CONNECT pers1int WITH iroutine1;
  IPers counter, pers1int;
  ...
  IDelete pers1int;
ENDPROC

TRAP iroutine1
  TPWrite "Current value of counter = " \Num:=counter;
ENDTRAP
```

Přikazuje přerušení, které se objeví vždy, když je změněno `counter` proměnné perzistentu. Potom je provedeno volání k trap rutině `iroutine1`.

Argumenty

```
IPers Name Interrupt
```

`Name`

Datový typ: `anytype`

Proměnná perzistentu, která bude definovat přerušení.

Všechny typy dat by mohly být použity, jako atomická, záznam, komponent záznamu, pole nebo prvek pole.

`Interrupt`

Datový typ: `intnum`

Identita přerušení. Předtím by mělo dojít k připojení k trap rutině prostřednictvím instrukce `CONNECT`.

Vykonávání programu

Když proměnná perzistentu mění hodnotu, je provedeno volání k odpovídající trap rutině. Když je tato rutina provedena, vykonávání programu pokračuje od místa vzniku přerušení.

Jestliže proměnná perzistentu mění hodnotu během zastavení programu, neobjeví se žádné přerušení při novém spuštění programu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Stejnou proměnnou pro identitu přerušení není možné používat vícekrát než jednou, bez toho, že by byla nejdříve vymazána. Viz instrukce - `ISignalDI`.

Jestliže jsou objednána data jako komponent záznamu nebo prvek pole určená v parametru `Name`, přerušení se objeví vždy, když je změněna jakákoliv část dat.

Při vykonávání trap rutiny a čtení hodnoty perzistentu není záruka, že přečtená hodnota je ta, která spustila přerušení.

Syntaxe

```
IPers
  [ Name ':=' ] < persistent (PERS) of anytype > ','
  [ Interrupt' :=' ] < variable (VAR) of intnum > ';'
```

Související informace

Pro informace o	Viz
Souhrn přerušení a správa přerušení	*Technická referenční příručka - Přehled RAPID*
Přerušení od vstupního signálu	*ISignalDI - Přikazuje přerušení od digitálního vstupního signálu na str 306*
Identita přerušení	*intnum - Identita přerušení na str 1516*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.117 IRMQMessage - Přikazuje přerušení RMQ pro datový typ

Použití

`IRMQMessage` (*Interrupt RAPID Message Queue Message*) se používá pro přikázání a zapnutí přerušení u konkrétního datového typu při používání funkce RMQ.

Základní příklady

Následující příklad názorně ukazuje instrukci `IRMQMessage`:

Viz také *IRMQMessage - Přikazuje přerušení RMQ pro datový typ na str 288*.

Příklad 1

```
VAR intnum rmqint;
VAR string dummy;
...
PROC main()
  CONNECT rmqint WITH iroutine1;
  IRMQMessage dummy, rmqint;
```

Přikazuje přerušení, které se objeví vždy, když je přijata nová `rmqmessage` obsahující datový typ `string`. Potom je provedeno volání k TRAP rutině `iroutine1`.

Argumenty

```
IRMQMessage InterruptDataType Interrupt
```

`InterruptDataType`

Datový typ: `anytype`

Reference k proměnné, perzistentu nebo konstantě datového typu, který bude generovat přerušení, když je přijata `rmqmessage` s určeným datovým typem.

`Interrupt`

Datový typ: `intnum`

Identita přerušení. Předtím by mělo dojít k připojení k TRAP rutině prostřednictvím instrukce `CONNECT`.

Vykonávání programu

Když je přijata zpráva RMQ s určeným datovým typem, je provedeno volání k odpovídající TRAP rutině. Po provedení pokračuje vykonávání programu od místa, kde se přerušení objevilo.

Všechny zprávy obsahující data stejného datového typu, bez ohledu na počet rozměrů, budou ošetřeny stejným přerušením. Při používání různých rozměrů použijte `RMQGetMsgHeader` pro přizpůsobení.

Všechny zprávy obsahující data datového typu, ke kterému není připojeno žádné přerušení, vyvolají varování.

Instrukce `RMQSendWait` má nejvyšší prioritu, jestliže zpráva je přijata a souhlasí s popisem očekávané odpovědi a zprávy připojené k TRAP rutině s instrukcí `IRMQMessage`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Všechny datové typy není možné používat v argumentu `InterruptDataType` (viz omezení).

Přerušení je považováno za bezpečné přerušení. Bezpečné přerušení nemůže být uloženo ke spánku s instrukcí `ISleep`. Událost bezpečného přerušení bude umístěna do fronty při zastavení programu a krokovém vykonávání, a při spuštění znovu v plynulém režimu bude přerušení vykonáno. Jediným časem, kdy bude bezpečné přerušení vyhozeno, je zaplnění fronty přerušení. Potom bude hlášena chyba. Přerušení nepřežije reset programu, např. PP na main.

Další příklady

Více příkladů jak používat instrukci `IRMQMessage` je názorně uvedeno dole.

Příklad 1

```
MODULE ReceiverMod
  VAR intnum intno1;
  VAR rmqheader rmqheader1;
  VAR rmqslot rmqslot1;
  VAR rmqmessage rmqmessage1;

  PROC main()
    VAR string interrupt_on_str := stEmpty;
    CONNECT intno1 WITH RecMsgs;
    ! Set up interrupts for data type string
    IRMQMessage interrupt_on_str, intno1;

    ! Perform cycle
    WHILE TRUE DO
      ...
    ENDWHILE
  ENDPROC
  TRAP RecMsgs
    VAR string receivestr;
    VAR string client_name;
    VAR num userdef;

    ! Get the message from the RMQ
    RMQGetMessage rmqmessage1;
    ! Get information about the message
    RMQGetMsgHeader rmqmessage1 \Header:=rmqheader1
         \SenderId:=rmqslot1 \UserDef:=userdef;

    IF rmqheader1.datatype = "string" AND rmqheader1.ndim = 0 THEN
      ! Get the data received in rmqmessage1
      RMQGetMsgData rmqmessage1, receivestr;
      client_name := RMQGetSlotName(rmqslot1);
      TPWrite "Rec string: " + receivestr;
      TPWrite "User Def: " + ValToStr(userdef);
      TPWrite "From: " + client_name;
    ELSE
      TPWrite "Faulty data received!"
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.117 IRMQMessage - Přikazuje přerušení RMQ pro datový typ

FlexPendant Interface, PC Interface, or Multitasking

Pokračování

```
    ENDIF

  ENDTRAP
ENDMODULE
```

Příklad ukazuje, jak nastavit přerušení pro určitý datový typ. Když je přijata zpráva, je vykonáno `TRAPRecMsgs` a přijatá data ve zprávě jsou vytištěna na FlexPendant. Jestliže přijatý datový typ nebo rozměr dat je odlišný od očekávaného, je to vytištěno na FlexPendant.

Omezení

Není dovoleno vykonávat `IRMQMessage` v synchronizovaném režimu. Způsobí to fatální chybu při běhu.

Není možné nastavovat přerušení, odesílat nebo přijímat datové instance datových typů, které jsou nehodnotové, polohodnotové typy nebo datových typů `motsetdata`.

Stejná proměnná pro identitu přerušení se nemůže použít více než jednou, aniž by ta první byla vymazána. Přerušení by se tady měla ošetřovat tak, jak je ukázáno na jedné z alternativ dole.

```
VAR intnum rmqint;
PROC main ()
  VAR mytype dummy;
  CONNECT rmq1int WITH iroutine1;
  IRMQMessage dummy, rmqint;
  WHILE TRUE DO
  ...
  ENDWHILE
ENDPROC
```

Veškerá aktivace přerušení se provádí na začátku programu. Tyto počáteční instrukce jsou potom drženy mimo hlavní tok programu.

```
VAR intnum rmqint;
PROC main ( )
  VAR mytype dummy;
  CONNECT rmqint WITH iroutine1;
  IRMQMessage dummy, rmqint;
  ...
  IDelete rmqint;
ENDPROC
```

Přerušení je vymazáno na konci programu a je potom znovu aktivováno. Všimněte si v tomto případě, že přerušení je po krátkou dobu neaktivní.

Syntaxe

```
IRMQMessage
  [ InterruptDataType ':=' ] < reference (REF) of anytype >
  [ Interrupt ':=' ] < variable (VAR) of intnum >';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*
Odeslat data do fronty úlohy RAPID nebo klienta Robot Application Builder	*RMQFindSlot - Najít identitu slotu od jména slotu na str 554*
Získat první zprávu z fronty zpráv RAPID (RMQ).	*RMQGetMessage - Získat zprávu RMQ na str 556*
Odeslat data do fronty úlohy RAPID nebo klienta Robot Application Builder a čekat na odpověď od klienta.	*RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu na str 572*
Vyjímá hlavičku dat z `rmqmessage`.	*RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ na str 562*
Odeslat data do fronty úlohy RAPID nebo klienta Robot Application Builder	*RMQSendMessage - Odeslat datovou zprávu RMQ na str 568*
Vyjímá data z `rmqmessage`.	*RMQGetMsgData - Získat datovou část zprávy RMQ na str 559*
Získat jméno slotu od určené identity slotu.	*RMQGetSlotName - Získat jméno klienta RMQ na str 1302*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.118 ISignalAI - Přerušuje od analogového vstupního signálu

Použití

`ISignalAI` (*Interrupt Signal Analog Input*) se používá pro přikázání a zapnutí přerušení od analogového vstupního signálu.

Základní příklady

Následující příklady názorně ukazují instrukci `ISignalAI`:

Příklad 1

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalAI \Single, ai1, AIO_BETWEEN, 1.5, 0.5, 0, sig1int;
```

Přikazuje přerušení, které se objeví poprvé, když logická hodnota analogového vstupního signálu `ai1` je mezi `0.5` a `1.5`. Potom je provedeno volání trap rutiny `iroutine1`.

Příklad 2

```
ISignalAI ai1, AIO_BETWEEN, 1.5, 0.5, 0.1, sig1int;
```

Přikazuje přerušení, které se objeví pokaždé, když logická hodnota analogového vstupního signálu `ai1` je mezi `0.5` a `1.5`, a absolutní rozdíl signálu v porovnání s uloženou referenční hodnotou je větší než `0.1`.

Příklad 3

```
ISignalAI ai1, AIO_OUTSIDE, 1.5, 0.5, 0.1, sig1int;
```

Přikazuje přerušení, které se objeví pokaždé, když logická hodnota analogového vstupního signálu `ai1` je nižší než 0,5 a vyšší než 1,5 a absolutní rozdíl signálu v porovnání s uloženou referenční hodnotou je větší než 0,1.

Argumenty

```
ISignalAI [\Single] | [\SingleSafe] Signal Condition HighValue
     LowValue DeltaValue [\DPos] | [\DNeg] Interrupt
```

`[\Single]`

Datový typ: `switch`

Určuje, jestli se přerušení má objevit jednou nebo cyklicky. Jestliže je nastaven argument `Single`, přerušení se objevuje nanejvýš jednou. Jestliže argumenty `Single` a `SingleSafe` jsou vypuštěny, přerušení se objeví pokaždé, když je jeho podmínka splněna.

`[\SingleSafe]`

Datový typ: `switch`

Určuje, že přerušení je jednoduché a bezpečné. Pro definici jednoduchého se podívejte na popis argumentu `Single`. Bezpečné přerušení nemůže být uloženo ke spánku s instrukcí `ISleep`. Událost bezpečného přerušení bude umístěna do fronty při zastavení programu a krokovém vykonávání, a při spuštění znovu v plynulém režimu bude přerušení vykonáno. Jediným časem, kdy bude bezpečné

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

přerušení vyhozeno, je zaplnění fronty přerušení. Potom bude hlášena chyba. Přerušení nepřežije reset programu, např. PP na main.

`Signal`

Datový typ: `signalai`

Jméno signálu, který bude generovat přerušení.

`Condition`

Datový typ: `aiotrigg`

Určuje, jak `HighValue` a `LowValue` definují podmínku, která má být splněna:

Hodnota	Symbolická konstanta	Komentář
1	`AIO_ABOVE_HIGH`	Signál bude generovat přerušení, jestliže je nad určenou vysokou hodnotou
2	`AIO_BELOW_HIGH`	Signál bude generovat přerušení, jestliže je pod určenou hodnotou
3	`AIO_ABOVE_LOW`	Signál bude generovat přerušení, jestliže je nad určenou nízkou hodnotou
4	`AIO_BELOW_LOW`	Signál bude generovat přerušení, jestliže je pod určenou nízkou hodnotou
5	`AIO_BETWEEN`	Signál bude generovat přerušení, jestliže je mezi určenou nízkou a vysokou hodnotou
6	`AIO_OUTSIDE`	Signál bude generovat přerušení, jestliže je pod určenou nízkou hodnotou a nad určenou vysokou hodnotou
7	`AIO_ALWAYS`	Signál bude vždy generovat přerušení

`HighValue`

Datový typ: `num`

Vysoká logická hodnota pro definování podmínky.

`LowValue`

Datový typ: `num`

Nízká logická hodnota pro definování podmínky.

`DeltaValue`

Datový typ: `num`

Definuje min rozdíl logického signálu před generováním nového přerušení. Hodnota aktuálního signálu ve srovnání s uloženou referenční hodnotou musí být větší než určená `DeltaValue` před generováním nového přerušení.

`[\DPos]`

Datový typ: `switch`

Určuje, že pouze kladné rozdíly logických signálů budou dávat nová přerušení.

`[\DNeg]`

Datový typ: `switch`

Určuje, že pouze záporné rozdíly logických signálů budou dávat nová přerušení.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.118 ISignalAI - Přerušuje od analogového vstupního signálu
RobotWare - OS
Pokračování

Jestliže není použit žádný z argumentů \DPos a \DNeg, kladné i záporné rozdíly budou generovat nová přerušení.

Interrupt

Datový typ: intnum

Identita přerušení. Předtím by mělo dojít k připojení přerušení k trap rutině prostřednictvím instrukce CONNECT.

Vykonávání programu

Když signál splňuje určené podmínky (Condition a DeltaValue), je provedeno volání k odpovídající trap rutině. Po provedení pokračuje vykonávání programu od místa, kde se přerušení objevilo.

Podmínky pro vznik přerušení

Předtím, než je přikázáno objednání přerušení, signál je pokaždé vzorkován, hodnota signálu je přečtena, uložena a později použita jako referenční hodnota pro podmínku DeltaValue.

V čase objednání přerušení, jestliže určená DeltaValue = 0 a po čase objednání přerušení je signál vzorkován. Hodnota signálu je potom porovnána s HighValue a LowValue podle Condition a s ohledem na DeltaValue, aby bylo rozhodnuto, jestli by mělo být generováno přerušení nebo nikoliv. Jestliže nově přečtená hodnota vyhovuje určené HighValue a LowValueCondition, ale její rozdíl v porovnání s naposledy uloženou referenční hodnotou je menší nebo se rovná argumentu DeltaValue, žádné přerušení nevznikne. Jestliže rozdíl signálu není v určeném směru, žádné přerušení nevznikne (argument \DPos or \DNeg).

Uložená referenční hodnota pro podmínku DeltaValue se aktualizuje s nově přečtenou hodnotou pro použití později u jakéhokoliv vzorku, jestliže jsou splněny následující podmínky:

- Argument Condition s určenou HighValue a LowValue (v rámci limitů)
- Argument DeltaValue (změna dostatečného signálu v jakémkoliv směru nezávisle na určeném přepínači \DPos nebo \DNeg)

Referenční hodnota se aktualizuje pouze v čase vzorku, nikoliv v čase objednání přerušení.

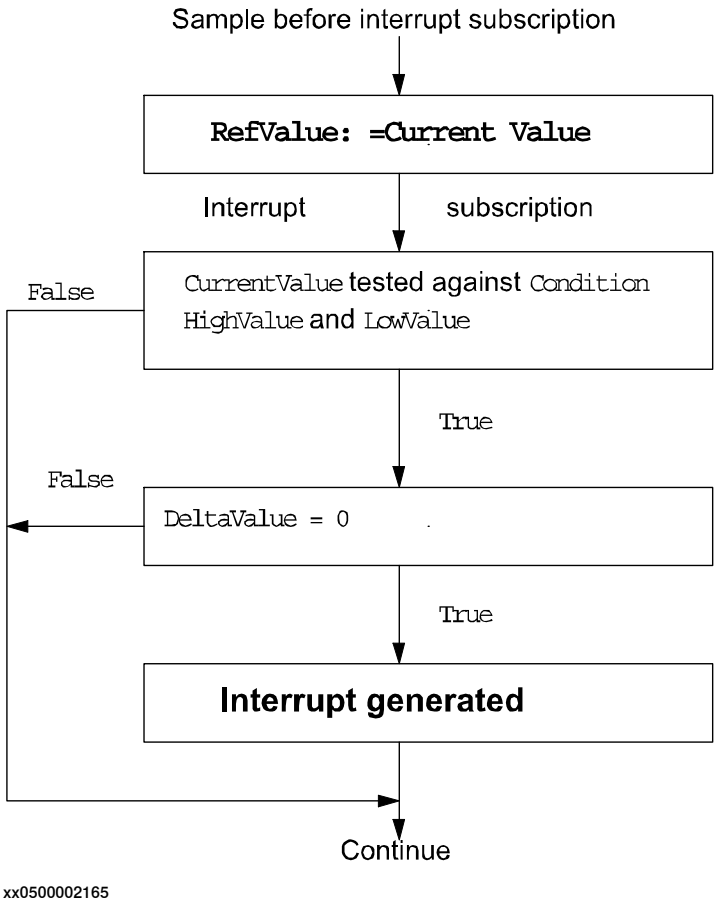
Přerušení je také generováno u vzorku pro aktualizaci referenční hodnoty, jestliže směr rozdílu signálu je v souladu s určeným argumentem (jakýkoliv směr, \DPos0, nebo \DNeg).

Když se používá přepínač \Single, bude generováno nejvýše jedno přerušení. Jestliže přepínač \Single (cyklické přerušení) není použit, je proveden nový test určených podmínek (Condition a DeltaValue) u každého vzorku hodnoty signálu. Je provedeno srovnání mezi hodnotou aktuálního signálu a naposledy uloženou referenční hodnotou, aby mohlo být rozhodnuto, jestli má být generováno přerušení nebo nikoliv.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Podmínka pro generování přerušení v čase objednání přerušení



Pokračování na další straně

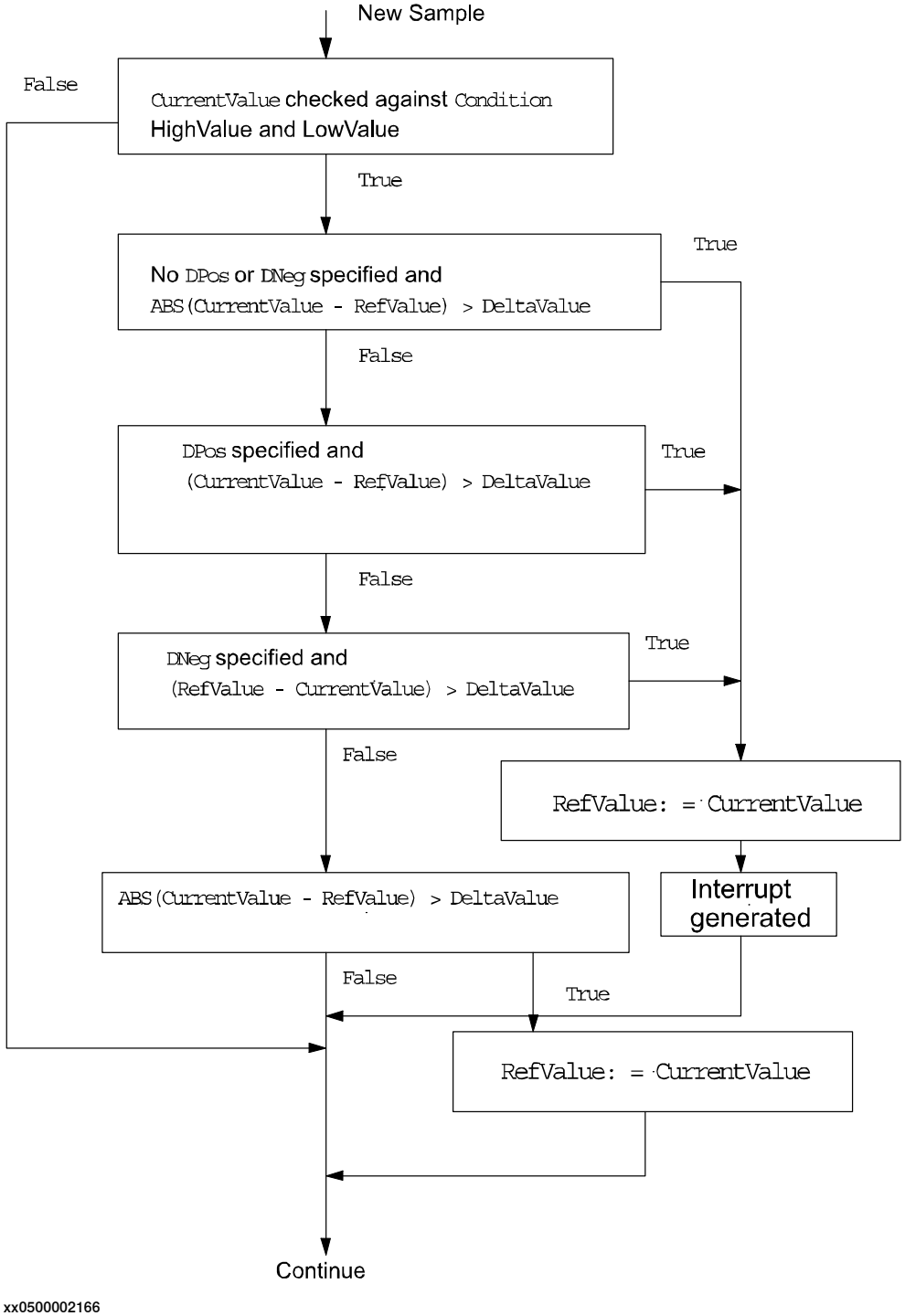
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.118 ISignalAI - Přerušuje od analogového vstupního signálu

RobotWare - OS

Pokračování

Podmínka pro generování přerušení u každého vzorku po objednání přerušení

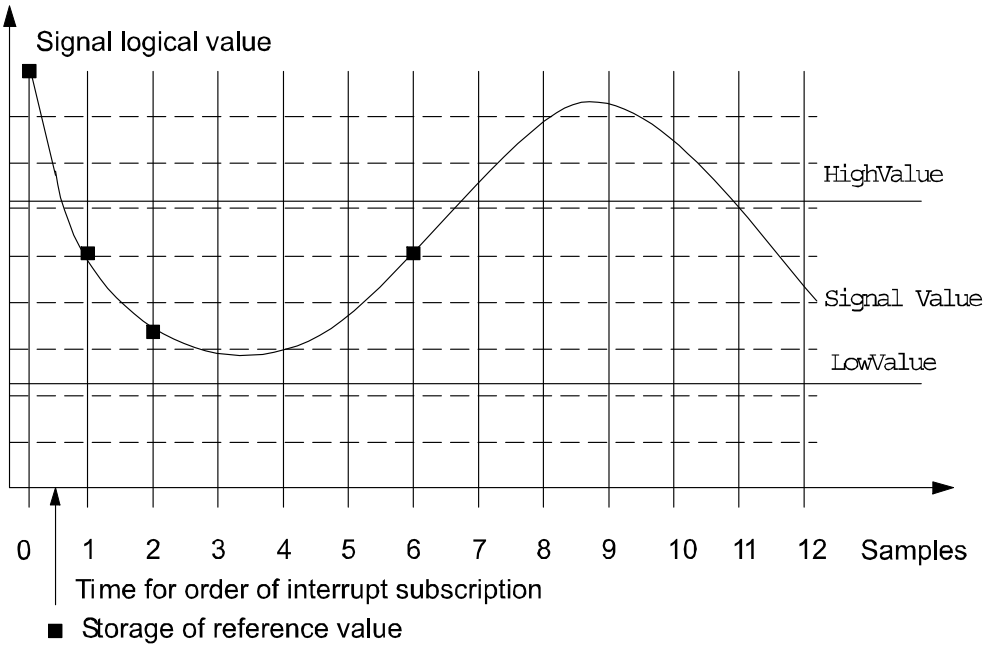


xx0500002166

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Příklad 1 generování přerušení



xx0500002167

Při předpokladu, že přerušení je přikázáno mezi vzorkem 0 a 1, následující instrukce poskytne následující výsledky:

```
ISignalAI ai1, AIO_BETWEEN, 6.1, 2.2, 1.0, sig1int;
```

Vzorek 1 bude generovat přerušení, protože hodnota signálu je mezi `HighValue` **a** `LowValue` **a rozdíl signálu ve srovnání se vzorkem 0 je více než** `DeltaValue`**.**

Vzorek 2 bude generovat přerušení, protože hodnota signálu je mezi `HighValue` **a** `LowValue` **a rozdíl signálu ve srovnání se vzorkem 1 je více než** `DeltaValue`**.**

Vzorky 3, 4, 5 nebudou generovat žádné přerušení, protože rozdíl signálu je méně než `DeltaValue`**.**

Vzorek 6 bude generovat přerušení.

Vzorky 7 až 10 nebudou generovat žádné přerušení, protože signál je nad `HighValue`**.**

Vzorek 11 nebude generovat žádné přerušení, protože rozdíl signálu ve srovnání se vzorkem 6 je rovný `DeltaValue`**.**

Vzorek 12 nebude generovat žádné přerušení, protože rozdíl signálu ve srovnání se vzorkem 6 je méně než `DeltaValue`**.**

Pokračování na další straně

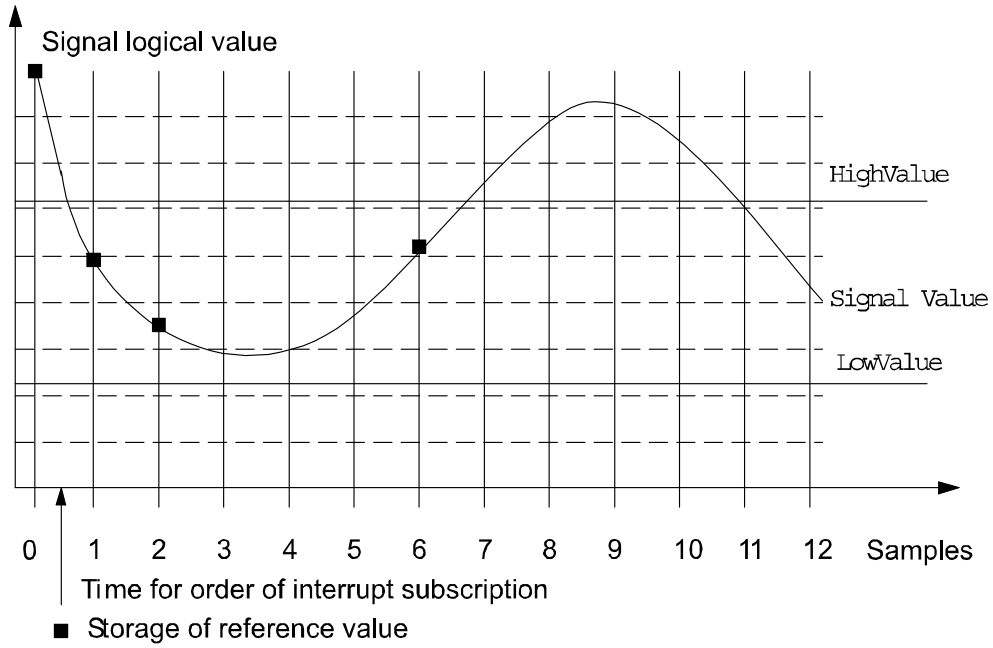
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.118 ISignalAI - Přerušuje od analogového vstupního signálu

RobotWare - OS

Pokračování

Příklad 2 generování přerušení



xx0500002168

Při předpokladu, že přerušení je přikázáno mezi vzorkem 0 a 1, následující instrukce poskytne následující výsledky:

```
ISignalAI ai1, AIO_BETWEEN, 6.1, 2.2, 1.0 \DPos, sig1int;
```

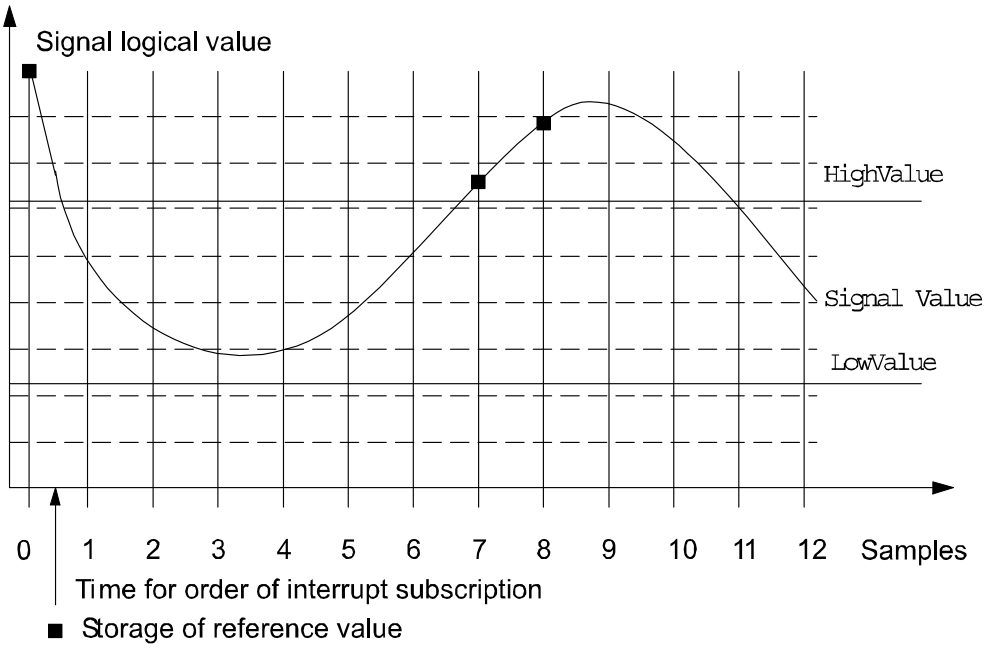
Nová referenční hodnota je uložena jako vzorek 1 a 2, protože signál je v limitech a absolutní rozdíl signálu mezi aktuální hodnotou a naposledy uloženou referenční hodnotu je větší než 1.0. Nebude generováno žádné přerušení, protože změny signálu jsou v záporném směru.

Vzorek 6 bude generovat přerušení, protože hodnota signálu je mezi `HighValue` **a** `LowValue` **a rozdíl signálu v kladném směru ve srovnání se vzorkem 2 je více než** `DeltaValue`**.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Příklad 3 generování přerušení



xx0500002169

Při předpokladu, že přerušení je přikázáno mezi vzorkem 0 a 1, následující instrukce poskytne následující výsledky:

```
ISignalAI \Single, ai1, AIO_OUTSIDE, 6.1, 2.2, 1.0 \DPos, sig1int;
```

Nová referenční hodnota je uložena jako vzorek 7, protože signál je v limitech a absolutní rozdíl signálu mezi aktuální hodnotou a naposledy uloženou referenční hodnotu je větší než 1.0.

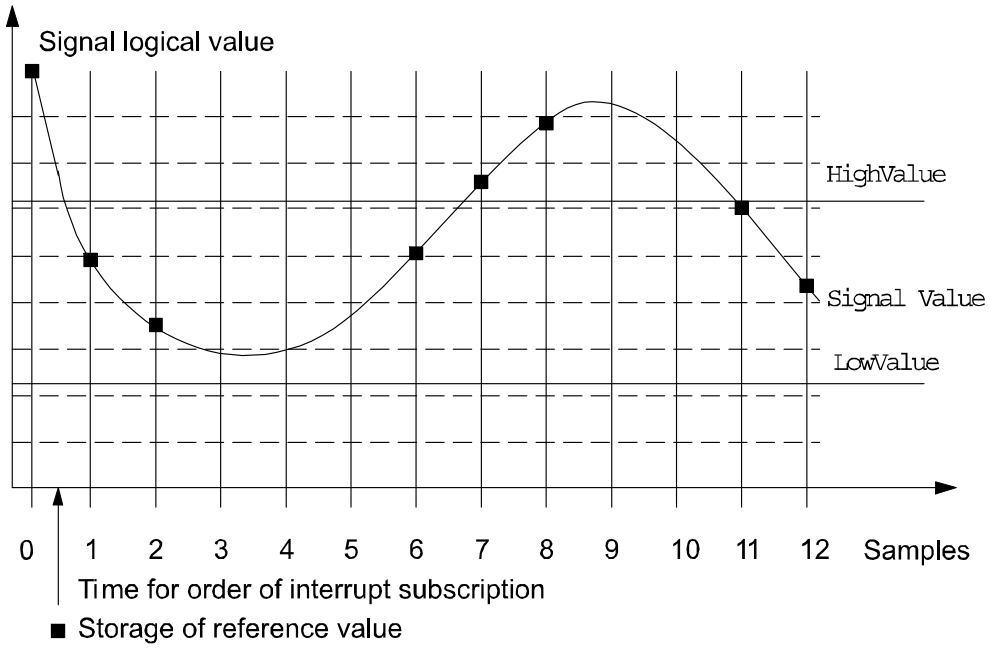
Vzorek 8 bude generovat přerušení, protože hodnota signálu je nad `HighValue` **a HighValue a rozdíl signálu v kladném směru ve srovnání se vzorkem 7 je více než** `DeltaValue`**.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.118 ISignalAI - Přerušuje od analogového vstupního signálu
RobotWare - OS
Pokračování

Příklad 4 generování přerušení



xx0500002170

Při předpokladu, že přerušení je přikázáno mezi vzorkem 0 a 1, následující instrukce poskytne následující výsledky:

```
ISignalAI ai1, AIO_ALWAYS, 6.1, 2.2, 1.0 \DPos, sig1int;
```

Nová referenční hodnota je uložena jako vzorek 1 a 2, protože signál je v limitech a absolutní rozdíl signálu mezi aktuální hodnotou a naposledy uloženou referenční hodnotu je větší než 1.0.

Vzorek 6 bude generovat přerušení, protože rozdíl signálu v kladném směru ve srovnání se vzorkem 2 je více než `DeltaValue`.

Vzorek 7 a 8 bude generovat přerušení, protože rozdíl signálu v kladném směru ve srovnání s předchozím vzorkem je více než `DeltaValue`.

Nová referenční hodnota je uložena jako vzorek 11 a 12, protože signál je v limitech a absolutní rozdíl signálu mezi aktuální hodnotou a naposledy uloženou referenční hodnotu je větší než 1.0.

Řešení chyb

Jestliže existuje objednání přerušení na analogovém vstupním signálu, přerušení bude dáno pro každou změnu v analogové hodnotě, která splňuje podmínku určenou při přikazování objednání přerušení. Jestliže analogová hodnota má šum, může být generováno mnoho přerušení, i když pouze jeden nebo dva bity v analogové hodnotě jsou změněny.

Aby se předešlo generování přerušení kvůli malým změnám hodnoty analogového vstupu, nastavte `DeltaValue` na úroveň větší než 0. Potom nebudou generována žádná přerušení, dokud změna analogové hodnoty nebude větší než určená `DeltaValue`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_NO_ALIASIO_DEF	Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.
ERR_AO_LIM	Naprogramovaný argument HighValue nebo LowValue pro určený analogový vstupní signál Signal je mimo limity.
ERR_NORUNUNIT	Není žádný kontakt s jednotkou I/O.

Omezení

Argumenty HighValue a LowValue by měly být v rozsahu: logická max hodnota, logická min hodnota definovaná pro signál.

HighValue musí být nad LowValue.

DeltaValue musí být 0 nebo kladná.

Omezení pro identitu přerušení jsou stejná jako pro ISignalDI.

Syntaxe

```
ISignalAI
  [ '\' Single ] | [ '\' SingleSafe ] ','
  [ Signal ':=' ] <variable (VAR) of signalai> ','
  [ Condition ':=' ] <expression (IN) of aiotrigg> ','
  [ HighValue ':=' ] <expression (IN) of num> ','
  [ LowValue ':=' ] <expression (IN) of num> ','
  [ DeltaValue ':=' ] <expression (IN) of num>
  [['\'DPos] | [ '\' DNeg] ',']
  [ Interrupt ':=' ] <variable (VAR) of intnum> ';'
```

Související informace

Pro informace o	Viz
Souhrn přerušení a správa přerušení	*Technická referenční příručka - Přehled RAPID*
Definice konstant	*aiotrigg - Analogová I/O trigger podmínka na str 1437*
Přerušení od analogového výstupního signálu	*ISignalAO - Přerušuje od analogového výstupního signálu na str 302*
Přerušení od digitálního vstupního signálu	*ISignalDI - Přikazuje přerušení od digitálního vstupního signálu na str 306*
Přerušení od digitálního výstupního signálu	*ISignalDO - Přerušuje od digitálního výstupního signálu na str 309*
Identita přerušení	*intnum - Identita přerušení na str 1516*
Související systémové parametry (filtr)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.119 ISignalAO - Přerušuje od analogového výstupního signálu

Použití

`ISignalAO` (*Interrupt Signal Analog Input*) se používá pro přikázání a zapnutí přerušení od analogového vstupního signálu.

Základní příklady

Následující příklady názorně ukazují instrukci `ISignalAO`:

Příklad 1

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalAO \Single, ao1, AIO_BETWEEN, 1.5, 0.5, 0, sig1int;
```

Přikazuje přerušení, které se objeví poprvé, když logická hodnota analogového výstupního signálu `ao1` je mezi `0.5` a `1.5`. Potom je provedeno volání trap rutiny `iroutine1`.

Příklad 2

```
ISignalAO ao1, AIO_BETWEEN, 1.5, 0.5, 0.1, sig1int;
```

Přikazuje přerušení, které se objeví pokaždé, když logická hodnota analogového výstupního signálu `ao1` je mezi `0.5` a `1.5`, a absolutní rozdíl signálu v porovnání s předchozí uloženou referenční hodnotou je větší než 0,1.

Příklad 3

```
ISignalAO ao1, AIO_OUTSIDE, 1.5, 0.5, 0.1, sig1int;
```

Přikazuje přerušení, které se objeví pokaždé, když logická hodnota analogového výstupního signálu `ao1` je nižší než `0.5` nebo vyšší než `1.5`, a absolutní rozdíl signálu v porovnání s předchozí uloženou referenční hodnotou je větší než 0,1.

Argumenty

```
ISignalAO [\Single] | [\SingleSafe] Signal Condition HighValue
     LowValue DeltaValue [\DPos] | [\DNeg] Interrupt
```

`[\Single]`

Datový typ: `switch`

Určuje, jestli se přerušení má objevit jednou nebo cyklicky. Jestliže je nastaven argument `Single`, přerušení se objevuje nanejvýš jednou. Jestliže argumenty `Single` a `SingleSafe` jsou vypuštěny, přerušení se objeví pokaždé, když je jeho podmínka splněna.

`[\SingleSafe]`

Datový typ: `switch`

Určuje, že přerušení je jednoduché a bezpečné. Pro definici jednoduchého se podívejte na popis argumentu `Single`. Bezpečné přerušení nemůže být uloženo ke spánku s instrukcí `ISleep`. Událost bezpečného přerušení bude umístěna do fronty při zastavení programu a krokovém vykonávání, a při spuštění znovu v plynulém režimu bude přerušení vykonáno. Jediným časem, kdy bude bezpečné

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

přerušení vyhozeno, je zaplnění fronty přerušení. Potom bude hlášena chyba. Přerušení nepřežije reset programu, např. PP na main.

`Signal`

Datový typ: `signalao`

Jméno signálu, který bude generovat přerušení.

`Condition`

Datový typ: `aiotrigg`

Určuje, jak `HighValue` a `LowValue` definují podmínku, která má být splněna:

Hodnota	Symbolická konstanta	Komentář
1	`AIO_ABOVE_HIGH`	Signál bude generovat přerušení, jestliže je nad určenou vysokou hodnotou
2	`AIO_BELOW_HIGH`	Signál bude generovat přerušení, jestliže je pod určenou hodnotou
3	`AIO_ABOVE_LOW`	Signál bude generovat přerušení, jestliže je nad určenou nízkou hodnotou
4	`AIO_BELOW_LOW`	Signál bude generovat přerušení, jestliže je pod určenou nízkou hodnotou
5	`AIO_BETWEEN`	Signál bude generovat přerušení, jestliže je mezi určenou nízkou a vysokou hodnotou
6	`AIO_OUTSIDE`	Signál bude generovat přerušení, jestliže je pod určenou nízkou hodnotou a nad určenou vysokou hodnotou
7	`AIO_ALWAYS`	Signál bude vždy generovat přerušení

`HighValue`

Datový typ: `num`

Vysoká logická hodnota pro definování podmínky.

`LowValue`

Datový typ: `num`

Nízká logická hodnota pro definování podmínky.

`DeltaValue`

Datový typ: `num`

Definuje min rozdíl logického signálu před generováním nového přerušení. Hodnota aktuálního signálu ve srovnání s předchozí uloženou referenční hodnotou musí být větší než určená `DeltaValue` před generováním nového přerušení.

`[\DPos]`

Datový typ: `switch`

Určuje, že pouze kladné rozdíly logických signálů budou dávat nová přerušení.

`[\DNeg]`

Datový typ: `switch`

Určuje, že pouze záporné rozdíly logických signálů budou dávat nová přerušení.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.119 ISignalAO - Přerušuje od analogového výstupního signálu

RobotWare - OS

Pokračování

Jestliže není použit žádný z argumentů `\DPos` a `\DNeg`, kladné i záporné rozdíly budou generovat nová přerušení.

`Interrupt`

Datový typ: `intnum`

Identita přerušení. Předtím by mělo dojít k připojení přerušení k trap rutině prostřednictvím instrukce `CONNECT`.

Vykonávání programu

Viz instrukce `ISignalAI` s informacemi o:

- Vykonávání programu
- Podmínka pro vznik přerušení
- Další příklady

Stejné principy platí pro `ISignalAO` jako pro `ISignalAI`.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_NO_ALIASIO_DEF`	Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.
`ERR_AO_LIM`	Naprogramovaný argument `HighValue` nebo `LowValue` pro určený analogový vstupní signál `Signal` je mimo limity.
`ERR_NORUNUNIT`	Není žádný kontakt s jednotkou I/O.

Omezení

Argumenty `HighValue` a `LowValue` by měly být v rozsahu: logická max hodnota, logická min hodnota definovaná pro signál.

`HighValue` musí být nad `LowValue`.

`DeltaValue` musí být 0 nebo kladná.

Omezení pro identitu přerušení jsou stejná jako pro `ISignalDO`.

Syntaxe

```
ISignalAO
[ ’\’ Single ] | [ ’\’ SingleSafe ] ’,’
[ Signal’:=’ ]<variable (VAR) of signalao>’,’
[ Condition’:=’ ]<expression (IN) of aiotrigg>’,’
[ HighValue’:=’ ]<expression (IN) of num>’,’
[ LowValue’:=’ ]<expression (IN) of num>’,’
[ DeltaValue’:=’ ]<expression (IN) of num>
[[’\’DPos] | [ ’\’DNeg] ’,’]
[ Interrupt’:=’ ]<variable (VAR) of intnum>’;’
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Souhrn přerušení a správa přerušení	*Technická referenční příručka - Přehled RAPID*
Definice konstant	*aiotrigg - Analogová I/O trigger podmínka na str 1437*
Přerušení od analogového vstupního signálu	*ISignalAI - Přerušuje od analogového vstupního signálu na str 292*
Přerušení od digitálního vstupního signálu	*ISignalDI - Přikazuje přerušení od digitálního vstupního signálu na str 306*
Přerušení od digitálního výstupního signálu	*ISignalDO - Přerušuje od digitálního výstupního signálu na str 309*
Identita přerušení	*intnum - Identita přerušení na str 1516*
Související systémové parametry (filtr)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.120 ISignalDI - Přikazuje přerušení od digitálního vstupního signálu

Použití

`ISignalDI` (*Interrupt Signal Digital In*) se používá pro přikázání a zapnutí přerušení od digitálního vstupního signálu.

Základní příklady

Následující příklady názorně ukazují instrukci `ISignalDI`:

Příklad 1

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalDI di1,1,sig1int;
```

Přikazuje přerušení, které se objeví vždy, když je digitální vstupní signál `di1` nastaven na `1`. Potom je provedeno volání k trap rutině `iroutine1`.

Příklad 2

```
ISignalDI di1,0,sig1int;
```

Přikazuje přerušení, které se objeví vždy, když je digitální vstupní signál `di1` nastaven na `0`.

Příklad 3

```
ISignalDI \Single, di1,1,sig1int;
```

Přikazuje přerušení, které se objeví pouze poprvé, když je digitální vstupní signál `di1` nastaven na `1`.

Argumenty

```
ISignalDI [ \Single] | [ \SingleSafe] Signal TriggValue Interrupt
```

`[ \Single ]`

Datový typ: `switch`

Určuje, jestli se přerušení má objevit jednou nebo cyklicky.

Jestliže je nastaven argument `Single`, přerušení se objevuje nanejvýš jednou. Jestliže argumenty `Single` a `SingleSafe` jsou vypuštěny, přerušení se objeví pokaždé, když je jeho podmínka splněna.

`[ \SingleSafe ]`

Datový typ: `switch`

Určuje, že přerušení je jednoduché a bezpečné. Pro definici jednoduchého se podívejte na popis argumentu `Single`. Bezpečné přerušení nemůže být uloženo ke spánku s instrukcí `ISleep`. Událost bezpečného přerušení bude umístěna do fronty při zastavení programu a krokovém vykonávání, a při spuštění znovu v plynulém režimu bude přerušení vykonáno. Jediným časem, kdy bude bezpečné přerušení vyhozeno, je zaplnění fronty přerušení. Potom bude hlášena chyba. Přerušení nepřežije reset programu, např. PP na main.

`Signal`

Datový typ: `signaldi`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jméno signálu, který bude generovat přerušení.

`TriggValue`

Datový typ: `dionum`

Hodnota, na kterou musí být signál změněn, aby se objevilo přerušení.

Hodnota je určena jako 0 nebo 1 nebo jako symbolická hodnota (např. `high/low`). Signál je spuštěn na hraně při změně na 0 nebo 1.

`TriggValue` 2 nebo symbolická hodnota `edge` se mohou používat pro generování přerušení jak na kladném boku (0 -> 1), tak i na záporném boku (1 -> 0).

`Interrupt`

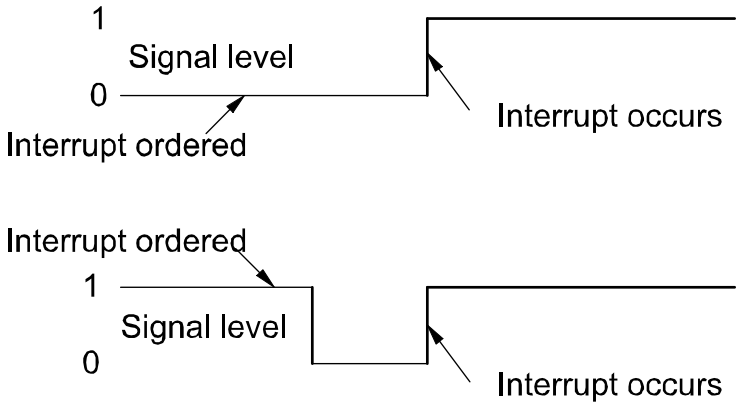
Datový typ: `intnum`

Identita přerušení. Předtím by mělo dojít k připojení k trap rutině prostřednictvím instrukce `CONNECT`.

Vykonávání programu

Když signál předpokládá určenou hodnotu, je provedeno volání k odpovídající trap rutině. Když je tato rutina provedena, vykonávání programu pokračuje od místa vzniku přerušení.

Když se signál změní na určenou hodnotu před přikázáním přerušení, neobjeví se žádné přerušení. Přerušení od digitálního vstupního signálu na signálové úrovni 1 je znázorněno na obrázku dole.



xx0500002189

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_NO_ALIASIO_DEF`	Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.
`ERR_NORUNUNIT`	Není žádný kontakt s jednotkou I/O.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.120 ISignalDI - Přikazuje přerušení od digitálního vstupního signálu

Omezení

Stejná proměnná pro identitu přerušení se nemůže použít více než jednou, aniž by ta první byla vymazána. Přerušení by se tady měla ošetřovat tak, jak je ukázáno na jedné z alternativ dole.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalDI di1, 1, sig1int;
  WHILE TRUE DO
  ...
  ENDWHILE
ENDPROC
```

Veškerá aktivace přerušení se provádí na začátku programu. Tyto počáteční instrukce jsou potom drženy mimo hlavní tok programu.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalDI di1, 1, sig1int;
  ...
  IDelete sig1int;
ENDPROC
```

Přerušení je vymazáno na konci programu a je potom znovu aktivováno. Všimněte si v tomto případě, že přerušení je po krátkou dobu neaktivní.

Syntaxe

```
ISignalDI
  [ '\' Single ] | [ '\' SingleSafe ] ',’
  [ Signal ':=' ] < variable (VAR) of signaldi > ',’
  [ TriggValue' :=' ] < expression (IN) of dionum > ',’
  [ Interrupt' :=' ] < variable (VAR) of intnum > ';’
```

Související informace

Pro informace o	Viz
Souhrn přerušení a správa přerušení	*Technická referenční příručka - Přehled RAPID*
Přerušení od výstupního signálu	*ISignalDO - Přerušuje od digitálního výstupního signálu na str 309*
Identita přerušení	*intnum - Identita přerušení na str 1516*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.121 ISignalDO - Přerušuje od digitálního výstupního signálu

Použití

`ISignalDO` (*Interrupt Signal Digital Out*) se používá pro přikázání a zapnutí přerušení od digitálního vstupního signálu.

Základní příklady

Následující příklady názorně ukazují instrukci `ISignalDO`:

Příklad 1

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalDO do1,1,sig1int;
```

Přikazuje přerušení, které se objeví vždy, když je digitální výstupní signál `do1` nastaven na `1`. Potom je provedeno volání k trap rutině `iroutine1`.

Příklad 2

```
ISignalDO do1,0,sig1int;
```

Přikazuje přerušení, které se objeví vždy, když je digitální výstupní signál `do1` nastaven na `0`.

Příklad 3

```
ISignalDO\Single, do1,1,sig1int;
```

Přikazuje přerušení, které se objeví pouze poprvé, když je digitální výstupní signál `do1` nastaven na `1`.

Argumenty

```
ISignalDO [ \Single ] | [ \SingleSafe ] Signal TriggValue Interrupt
```

`[ \Single ]`

Datový typ: `switch`

Určuje, jestli se přerušení má objevit jednou nebo cyklicky.

Jestliže je nastaven argument `Single`, přerušení se objevuje nanejvýš jednou. Jestliže argumenty `Single` a `SingleSafe` jsou vypuštěny, přerušení se objeví pokaždé, když je jeho podmínka splněna.

`[ \SingleSafe ]`

Datový typ: `switch`

Určuje, že přerušení je jednoduché a bezpečné. Pro definici jednoduchého se podívejte na popis argumentu `Single`. Bezpečné přerušení nemůže být uloženo ke spánku s instrukcí `ISleep`. Událost bezpečného přerušení bude umístěna do fronty při zastavení programu a krokovém vykonávání, a při spuštění znovu v plynulém režimu bude přerušení vykonáno. Jediným časem, kdy bude bezpečné přerušení vyhozeno, je zaplnění fronty přerušení. Potom bude hlášena chyba. Přerušení nepřežije reset programu, např. PP na main.

`Signal`

Datový typ: `signaldo`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.121 ISignalDO - Přerušuje od digitálního výstupního signálu
RobotWare - OS
Pokračování

Jméno signálu, který bude generovat přerušení.

`TriggValue`

Datový typ: `dionum`

Hodnota, na kterou musí být signál změněn, aby se objevilo přerušení.

Hodnota je určena jako 0 nebo 1 nebo jako symbolická hodnota (např. `high/low`). Signál je spuštěn na hraně při změně na 0 nebo 1.

`TriggValue` 2 nebo symbolická hodnota `edge` se mohou používat pro generování přerušení jak na kladném boku (0 -> 1), tak i na záporném boku (1 -> 0).

`Interrupt`

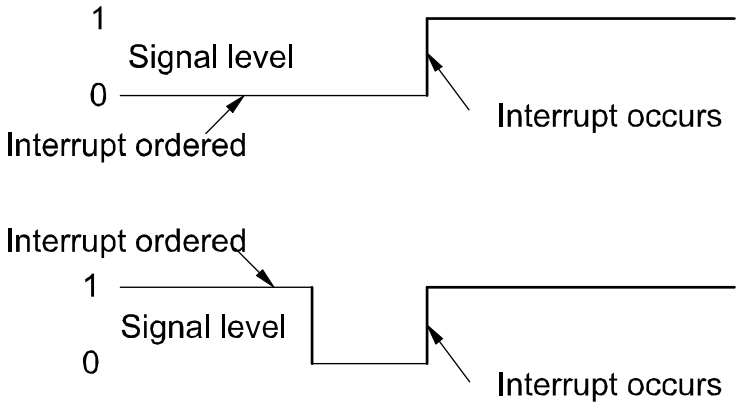
Datový typ: `intnum`

Identita přerušení. Předtím by mělo dojít k připojení k trap rutině prostřednictvím instrukce `CONNECT`.

Vykonávání programu

Když signál předpokládá určenou hodnotu 0 nebo 1, je provedeno volání k odpovídající trap rutině. Když je tato rutina provedena, vykonávání programu pokračuje od místa vzniku přerušení.

Když se signál změní na určenou hodnotu před přikázáním přerušení, neobjeví se žádné přerušení. Přerušení od digitálního výstupního signálu na signálové úrovni 1 je znázorněno na obrázku dole.



xx0500002190

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_NO_ALIASIO_DEF`	Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.
`ERR_NORUNUNIT`	Není žádný kontakt s jednotkou I/O.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Stejná proměnná pro identitu přerušení se nemůže použít více než jednou, aniž by ta první byla vymazána. Přerušení by se tady měla ošetřovat tak, jak je ukázáno na jedné z alternativ dole.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalDO do1, 1, sig1int;
  WHILE TRUE DO
  ...
  ENDWHILE
ENDPROC
```

Veškerá aktivace přerušení se provádí na začátku programu. Tyto počáteční instrukce jsou potom drženy mimo hlavní tok programu.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalDO do1, 1, sig1int;
  ...
  IDelete sig1int;
ENDPROC
```

Přerušení je vymazáno na konci programu a je potom znovu aktivováno. Všimněte si v tomto případě, že přerušení je po krátkou dobu neaktivní.

Syntaxe

```
ISignalDO
  [ '\' Single ] | [ '\' SingleSafe ] ','
  [ Signal ':=' ] < variable (VAR) of signaldo > ','
  [ TriggValue' :=' ] < expression (IN) of dionum > ','
  [ Interrupt' :=' ] < variable (VAR) of intnum > ';'
```

Související informace

Pro informace o	Viz
Souhrn přerušení a správa přerušení	*Technická referenční příručka - Přehled RAPID*
Přerušení od vstupního signálu	*ISignalDI - Přikazuje přerušení od digitálního vstupního signálu na str 306*
Identita přerušení	*intnum - Identita přerušení na str 1516*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.122 ISignalGI - Přikazuje přerušení od skupiny digitálních vstupních signálů

Použití

`ISignalGI` (*Interrupt Signal Group Digital In*) se používá pro přikázání a zapnutí přerušení od skupiny digitálních vstupních signálů.

Základní příklady

Následující příklad názorně ukazuje instrukci `ISignalGI`:

Příklad 1

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalGI gi1,sig1int;
```

Přikazuje přerušení, když skupinový digitální vstupní signál mění hodnotu.

Argumenty

```
ISignalGI [ \Single ] | [ \SingleSafe ] Signal Interrupt
```

`[ \Single ]`

Datový typ: `switch`

Určuje, jestli se přerušení má objevit jednou nebo cyklicky.

Jestliže je nastaven argument `Single`, přerušení se objevuje nanejvýš jednou. Jestliže argumenty `Single` a `SingleSafe` jsou vypuštěny, přerušení se objeví pokaždé, když je jeho podmínka splněna.

`[ \SingleSafe ]`

Datový typ: `switch`

Určuje, že přerušení je jednoduché a bezpečné. Pro definici jednoduchého se podívejte na popis argumentu `Single`. Bezpečné přerušení nemůže být uloženo ke spánku s instrukcí `ISleep`. Událost bezpečného přerušení bude umístěna do fronty při zastavení programu a krokovém vykonávání, a při spuštění znovu v plynulém režimu bude přerušení vykonáno. Jediným časem, kdy bude bezpečné přerušení vyhozeno, je zaplnění fronty přerušení. Potom bude hlášena chyba. Přerušení nepřežije reset programu, např. PP na main.

`Signal`

Datový typ: `signalgi`

Jméno skupinového vstupního signálu, který generuje přerušení.

`Interrupt`

Datový typ: `intnum`

Identita přerušení. Předtím by mělo dojít k připojení k trap rutině prostřednictvím instrukce `CONNECT`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Když skupinový signál mění hodnotu, je provedeno volání k odpovídající trap rutině. Když je tato rutina provedena, vykonávání programu pokračuje od místa vzniku přerušení.

Jestliže se signál změní před přikázáním přerušení, žádné přerušení nevznikne.

Když je digitální skupinový vstupní signál nastaven na hodnotu, může to generovat několik přerušení. Důvodem je, že změny individuálních bitů zahrnutých do skupinového signálu nejsou detekovány ve stejném čase systému robotu. Kvůli zabránění vícenásobným přerušením u změny jednoho skupinového signálu může být pro signál určen čas filtru.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_NO_ALIASIO_DEF	Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.
ERR_NORUNUNIT	Není žádný kontakt s jednotkou I/O.

Omezení

Max počet signálů, které mohou být použity pro skupinu, je 32.

Podmínka numerické hodnoty nemůže být použita v instrukci k určení, že přerušení by se mělo objevit při změnách na tuto konkrétní hodnotu. To musí být ošetřeno v uživatelském programu přečtením hodnoty skupinového signálu při vykonávání TRAP.

Přerušení jsou generována jako bitová přerušení, např. přerušení při změně jednoduchého digitálního vstupního signálu v rámci skupiny. Jestliže bity ve skupinovém signálu mění hodnotu s prodlevou mezi nastaveními, bude generováno několik přerušení. Nutná je znalost o fungování I/O desky, abychom získali správnou funkci při používání ISignalGI. Jestliže je generováno několik přerušení u skupinových vstupních nastavení, použijte místo toho ISignalDI na výběrových signálech, které jsou nastaveny po nastavení všech bitů ve skupinovém signálu.

Stejná proměnná pro identitu přerušení se nemůže použít více než jednou, aniž by ta první byla vymazána. Přerušení by se tady měla ošetřovat tak, jak je ukázáno na jedné z alternativ dole.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalGI gi1, sig1int;
  WHILE TRUE DO
  ...
  ENDWHILE
ENDPROC
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.122 ISignalGI - Přikazuje přerušení od skupiny digitálních vstupních signálů

RobotWare - OS

Pokračování

Veškerá aktivace přerušení se provádí na začátku programu. Tyto počáteční instrukce jsou potom drženy mimo hlavní tok programu.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalGI gi1, sig1int;
  ...
  IDelete sig1int;
ENDPROC
```

Přerušení je vymazáno na konci programu a je potom znovu aktivováno. Všimněte si v tomto případě, že přerušení je po krátkou dobu neaktivní.

Syntaxe

```
ISignalGI
  [ '\' Single ] | [ '\' SingleSafe ] ',' 
  [ Signal ':=' ] < variable (VAR) of signalgi > ','
  [ Interrupt':=' ] < variable (VAR) of intnum > ';'
```

Související informace

Pro informace o	Viz
Souhrn přerušení a správa přerušení	*Technická referenční příručka - Přehled RAPID*
Přerušení od vstupního signálu	*ISignalDI - Přikazuje přerušení od digitálního vstupního signálu na str 306*
Přerušení od skupinových výstupních signálů	*ISignalGO - Přikazuje přerušení od skupiny digitálních výstupních signálů na str 315*
Identita přerušení	*intnum - Identita přerušení na str 1516*
Čas filtru	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.123 ISignalGO - Přikazuje přerušení od skupiny digitálních výstupních signálů

Použití

`ISignalGO` (*Interrupt Signal Group Digital Out*) se používá pro přikázání a zapnutí přerušení od skupiny digitálních výstupních signálů.

Základní příklady

Následující příklad názorně ukazuje instrukci `ISignalGO`:

Příklad 1

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalGO go1,sig1int;
```

Přikazuje přerušení, když skupinový digitální výstupní signál mění hodnotu.

Argumenty

```
ISignalGO [ \Single ] | [ \SingleSafe ] Signal Interrupt
```

`[ \Single ]`

Datový typ: `switch`

Určuje, jestli se přerušení má objevit jednou nebo cyklicky.

Jestliže je nastaven argument `\Single`, přerušení se objevuje nanejvýš jednou. Jestliže argumenty `Single` a `SingleSafe` jsou vypuštěny, přerušení se objeví pokaždé, když je jeho podmínka splněna.

`[ \SingleSafe ]`

Datový typ: `switch`

Určuje, že přerušení je jednoduché a bezpečné. Pro definici jednoduchého se podívejte na popis argumentu `Single`. Bezpečné přerušení nemůže být uloženo ke spánku s instrukcí `ISleep`. Událost bezpečného přerušení bude umístěna do fronty při zastavení programu a krokovém vykonávání, a při spuštění znovu v plynulém režimu bude přerušení vykonáno. Jediným časem, kdy bude bezpečné přerušení vyhozeno, je zaplnění fronty přerušení. Potom bude hlášena chyba. Přerušení nepřežije reset programu, např. PP na main.

`Signal`

Datový typ: `signalgo`

Jméno skupinového výstupního signálu, který generuje přerušení.

`Interrupt`

Datový typ: `intnum`

Identita přerušení. Předtím by mělo dojít k připojení k trap rutině prostřednictvím instrukce `CONNECT`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.123 ISignalGO - Přikazuje přerušení od skupiny digitálních výstupních signálů
RobotWare - OS
Pokračování

Vykonávání programu

Když skupinový signál mění hodnotu, je provedeno volání k odpovídající trap rutině. Když je tato rutina provedena, vykonávání programu pokračuje od místa vzniku přerušení.

Jestliže se signál změní před přikázáním přerušení, žádné přerušení nevznikne.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_NO_ALIASIO_DEF	Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.
ERR_NORUNUNIT	Není žádný kontakt s jednotkou I/O.

Omezení

Max počet signálů, které mohou být použity pro skupinu, je 32.

Podmínka numerické hodnoty nemůže být použita v instrukci k určení, že přerušení by se mělo objevit při změnách na tuto konkrétní hodnotu. To musí být ošetřeno v uživatelském programu přečtením hodnoty skupinového signálu při vykonávání TRAP.

Stejná proměnná pro identitu přerušení se nemůže použít více než jednou, aniž by ta první byla vymazána. Přerušení by se tady měla ošetřovat tak, jak je ukázáno na jedné z alternativ dole.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalGO go1, sig1int;
  WHILE TRUE DO
  ...
  ENDWHILE
ENDPROC
```

Veškerá aktivace přerušení se provádí na začátku programu. Tyto počáteční instrukce jsou potom drženy mimo hlavní tok programu.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalGO go1, sig1int;
  ...
  IDelete sig1int;
ENDPROC
```

Přerušení je vymazáno na konci programu a je potom znovu aktivováno. Všimněte si v tomto případě, že přerušení je po krátkou dobu neaktivní.

Syntaxe

```
ISignalGO
  [ '\' Single ] | [ '\' SingleSafe ] ','
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
[ Signal ':=' ] < variable (VAR) of signalgo > ','
[ Interrupt':=' ] < variable (VAR) of intnum > ';'
```

Související informace

Pro informace o	Viz
Souhrn přerušení a správa přerušení	*Technická referenční příručka - Přehled RAPID*
Přerušení od výstupního signálu	*ISignalDO - Přerušuje od digitálního výstupního signálu na str 309*
Přerušení od skupinových vstupních signálů	*ISignalGI - Přikazuje přerušení od skupiny digitálních vstupních signálů na str 312*
Identita přerušení	*intnum - Identita přerušení na str 1516*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.124 ISleep - Deaktivuje přerušení

Použití

`ISleep`(*Interrupt Sleep*) se používá k dočasné deaktivaci individuálního přerušení.

Během doby deaktivace jsou všechna generovaná přerušení určeného typu vyhozena bez vykonání trap rutiny.

Základní příklady

Následující příklad názorně ukazuje instrukci `ISleep`.

Viz také *Další příklady na str 318*.

Příklad 1

```
ISleep sig1int;
```

Přerušení `sig1int` je deaktivováno.

Argumenty

```
ISleep Interrupt
```

`Interrupt`

Datový typ: `intnum`

Proměnná (identita přerušení) přerušení.

Vykonávání programu

Všechna generovaná přerušení určeného typu jsou vyhozena bez vykonání jakékoliv trap rutiny, dokud přerušení nebylo znovu aktivováno prostřednictvím instrukce `IWatch`. Přerušení, která jsou generována, zatímco `ISleep` je realizováno, jsou ignorována.

Další příklady

Více příkladů instrukce `ISleep` je názorně uvedeno dole.

Příklad 1

```
VAR intnum timeint;
PROC main()
  CONNECT timeint WITH check_serialch;
  ITimer 60, timeint;
  ...
  ISleep timeint;
  WriteBin ch1, buffer, 30;
  IWatch timeint;
  ...
TRAP check_serialch
  WriteBin ch1, buffer, 1;
  IF ReadBin(ch1\Time:=5) < 0 THEN
    TPWrite "The serial communication is broken";
    EXIT;
  ENDIF
ENDTRAP
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Komunikace přes sériový kanál ch1 je monitorována prostřednictvím přerušení, která jsou generována každých 60 sekund. Trap rutina kontroluje, jestli komunikace funguje. Přerušení nejsou povolena, když komunikace běží.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_UNKINO	Číslo přerušení je neznámé. Přerušení, která nebyla přikázána ani zapnuta, nejsou povolena.
ERR_INOISSAFE	Jestliže zkoušíte deaktivovat dočasně bezpečné přerušení s ISleep.

Syntaxe

```
ISleep
  [ Interrupt ':=' ] < variable (VAR) of intnum > ';'
```

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*
Zapínání přerušení	*IWatch - Aktivuje přerušení na str 325*
Vypínání všech přerušení	*IDisable - Vypíná přerušení na str 244*
Zrušení přerušení	*IDelete - Zruší přerušení na str 243*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.125 ITimer - Přikazuje časované přerušení

Použití

`ITimer` (*Interrupt Timer*) se používá pro přikázání a zapnutí časovaného přerušení.

Tuto instrukci je možné používat například ke kontrole statutu periferního vybavení jednou za minutu.

Základní příklady

Následující příklady názorně ukazují instrukci `ITimer`:

Viz také *Další příklady na str 321*.

Příklad 1

```
VAR intnum timeint;
PROC main()
  CONNECT timeint WITH iroutine1;
  ITimer 60, timeint;
```

Přikazuje přerušení, které se objeví cyklicky každých `60` sekund. Potom je provedeno volání k trap rutině `iroutine1`.

Příklad 2

```
ITimer \Single, 60, timeint;
```

Přikazuje přerušení, které se objeví jednou, po `60` sekundách.

Argumenty

```
ITimer [ \Single ] | [ \SingleSafe ] Time Interrupt
```

`[ \Single ]`

Datový typ: `switch`

Určuje, jestli se přerušení má objevit jednou nebo cyklicky.

Jestliže je nastaven argument `Single`, přerušení se objevuje pouze jednou. Jestliže argumenty `Single` a `SingleSafe` jsou vypuštěny, přerušení se objeví pokaždé v určeném čase.

`[ \SingleSafe ]`

Datový typ: `switch`

Stanoví, že přerušení je jednoduché a bezpečné. Pro definici jednoduchého viz popis argumentu `Single`. Bezpečné přerušení není možné uložit ke spánku s instrukcí `ISleep`. Událost bezpečného přerušení bude uložena do fronty při zastavení programu a krokovém vykonávání, a když bude opět provedeno spuštění v plynulém režimu, přerušení bude provedeno.

`Time`

Datový typ: `num`

Množství času, které musí uplynout, než se přerušení objeví.

Hodnota je určena v sekundách. Jestliže je nastaveno `Single` nebo `SingleSafe`, tentokrát nesmí být kratší než 0,01 sek. Odpovídající čas pro cyklická přerušení je 0,1 sek.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Interrupt`

Datový typ: `intnum`

Proměnná (identita přerušení) přerušení. Předtím by mělo dojít k připojení přerušení k trap rutině prostřednictvím instrukce `CONNECT`.

Vykonávání programu

Odpovídající trap rutina je automaticky volána v daném čase po přikázání přerušení. Po provedení pokračuje vykonávání programu od místa, kde k přerušení došlo.

Jestliže se přerušení objevuje cyklicky, je spuštěn nový výpočet času od místa, kde k přerušení došlo.

Další příklady

Více příkladů instrukce `ITimer` je názorně uvedeno dole.

Příklad 1

```
VAR intnum timeint;
PROC main()
  CONNECT timeint WITH check_serialch;
  ITimer 60, timeint;
  ...
TRAP check_serialch
  WriteBin ch1, buffer, 1;
  IF ReadBin(ch1\Time:=5) < 0 THEN
    TPWrite "The serial communication is broken";
    EXIT;
  ENDIF
ENDTRAP
```

Komunikace přes sériový kanál `ch1` je monitorována prostřednictvím přerušení, která jsou generována každých `60` sekund. Trap rutina kontroluje, jestli komunikace funguje. Pokud tomu tak není, vykonávání programu je ukončeno a objeví se chybová zpráva.

Omezení

Stejnou proměnnou pro identitu přerušení není možné používat vícekrát než jednou, bez toho, že by byla nejdříve vymazána. Viz instrukce - `ISignalDI`.

Syntaxe

```
ITimer
  [ '\' Single ] | [ '\' SingleSafe ] ','
  [ Time ':=' ] < expression (IN) of num >','
  [ Interrupt' :=' ] < variable (VAR) of intnum > ';'
```

Související informace

Pro informace o	Viz
Souhrn přerušení a správa přerušení	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.126 IVarValue - přikazuje přerušení hodnoty proměnné

Použití

`IVarValue` (*Interrupt Variable Value*) je používaná pro přikázání a zapnutí přerušení, když byla změněna hodnota proměnné získané přes rozhraní sériového senzoru.

Tuto instrukci je možné používat například pro získání objemu svaru nebo hodnot spáry od sledovače svarů.

Základní příklady

Následující příklad názorně ukazuje instrukci `IVarValue`:

Příklad 1

```
LOCAL PERS num
     adptVlt{25}:=[1,1.2,1.4,1.6,1.8,2,2.16667,2.33333,2.5,...];
LOCAL PERS num
     adptWfd{25}:=[2,2.2,2.4,2.6,2.8,3,3.16667,3.33333,3.5,...];
LOCAL PERS num
     adptSpd{25}:=10,12,14,16,18,20,21.6667,23.3333,25[,...];
LOCAL CONST num GAP_VARIABLE_NO:=11;
PERS num gap_value;
VAR intnum IntAdap;

PROC main()
  ! Setup the interrupt. The trap routine AdapTrp will be called
  ! when the gap variable with number ´GAP_VARIABLE_NO’ in the
  !sensor interface has been changed. The new value will be
  ! available in the PERS gp_value variable.
  ! Connect to the sensor device "sen1:" (defined in sio.cfg).
  SenDevice "sen1:";

  CONNECT IntAdap WITH AdapTrp;
  IVarValue "sen1:", GAP_VARIABLE_NO, gap_value, IntAdap;

  ! Start welding
  ArcL\On,*,v100,adaptSm,adaptWd,adaptWv,z10,tool\j\Track:=track;
  ArcL\On,*,v100,adaptSm,adaptWd,adaptWv,z10,tool\j\Track:=track;

ENDPROC

TRAP AdapTrap
  VAR num ArrInd;
  !Scale the raw gap value received
  ArrInd:=ArrIndx(gap_value);

  ! Update active welddata PERS variable ‘adaptWd’ with new data
  ! from the arrays of predefined parameter arrays. The scaled gap
  ! value is used as index in the voltage, wirefeed and
  ! speed arrays.
  adaptWd.weld_voltage:=adptVlt{ArrInd};
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
    adaptWd.weld_wirefeed:=adptWfd{ArrInd};
    adaptWd.weld_speed:=adptSpd{ArrInd};

    !Request a refresh of AW parameters using the new data i adaptWd
    ArcRefresh;

ENDTRAP
```

Argumenty

```
IVarValue device VarNo Value Interrupt [\Unit] [\DeadBand]
     [\ReportAtTool] [\SpeedAdapt] [\APTR]
```

`device`

Datový typ: `string`

Jméno I/O zařízení konfigurované v sio.cfg pro použitý senzor.

`VarNo`

Datový typ: `num`

Číslo proměnné, která bude pod dohledem.

`Value`

Datový typ: `num`

Proměnná `PERS`**, která bude držet novou hodnotu** `VarNo`**.**

`Interrupt`

Datový typ: `intnum`

Proměnná (identita přerušení) přerušení. Předtím by mělo dojít k připojení přerušení k trap rutině prostřednictvím instrukce `CONNECT`**.**

`[\Unit]`

Datový typ: `num`

Faktor škálování, kterým bude hodnota senzoru pro `VarNo` **vynásobena před kontrolou a před uložením do** `Value`**.**

`[\DeadBand]`

Datový typ: `num`

Jestliže hodnota pro `Varno`**, vrácená senzorem, je v rozpětí** `+/- DeadBand`**, není generováno žádné přerušení.**

`[\ReportAtTool]`

Datový typ: `switch`

Tento volitelný argument je dostupný pouze pro senzory dopředného (look-ahead) typu, například optické sledovací senzory. Argument určuje, že hodnota proměnné by neměla být vyhodnocena okamžitě, ale když TCP robotu dosáhne pozice, tj. výhled (look-ahead) je kompenzován.

`[\SpeedAdapt]`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.126 IVarValue - přikazuje přerušení hodnoty proměnné

Optical Tracking

Pokračování

`\SpeedAdapt` je škálovací faktor používaný pro změnu procesní rychlosti v instrukcích `Arc` a `Cap`. Je vynásoben hodnotou senzoru pro `VarNo` podle:

procesní rychlost = `\SpeedAdapt * value(VarNo)`

`[\APTR]`

Datový typ: `switch`

Určuje, že objednání proměnné by mělo být navázáno na sledovač v bodě, například WeldGuide, určený v argumentu `device`.

Vykonávání programu

Odpovídající trap rutina je automaticky volána v daném čase po přikázání přerušení. Po provedení pokračuje vykonávání programu od místa, kde k přerušení došlo.

Omezení

Stejnou proměnnou pro identitu přerušení není možné používat vícekrát než pětkrát, bez toho, že by byla nejdříve vymazána.

UPOZORNĚNÍ

Příliš vysoká frekvence přerušení bude zdržovat celé vykonávání RAPID.

Syntaxe

```
IVarValue
  [ device ':=' ] < expression (IN) of string> ','
  [ VarNo ':=' ] < expression (IN) of num > ','
  [ Value ':=' ] < persistent (PERS) of num > ','
  [ Interrupt ':=' ] < variable (VAR) of intnum > ','
  [ '\' Unit ':=' ] < expression (IN) of num > ','
  [ '\' DeadBand ':=' ] < expression (IN) of num > ','
  [ '\' ReportAtTool ] ','
  [ '\' SpeedAdapt ':=' ] < expression (IN) of num > ','
  [ '\' APTR ] ';'
```

Související informace

Pro informace o	Viz
Připojit k zařízení senzoru	*SenDevice - Připojit k zařízení senzoru na str 614*
Souhrn přerušení a správa přerušení	*Technická referenční příručka - Přehled RAPID*
Optické sledování	*Application manual - Continuous Application Platform*
Oblouk optického sledování	*Application manual - Arc and Arc Sensor*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.127 IWatch - Aktivuje přerušení

Použití

`IWatch`(*Interrupt Watch*) se používá pro aktivaci přerušení, které bylo předtím přikázáno, ale bylo deaktivováno s `ISleep`.

Základní příklady

Následující příklad názorně ukazuje instrukci `IWatch`:

Viz také *Další příklady na str 325*.

Příklad 1

```
IWatch sig1int;
```

Přerušení `sig1int`, které bylo předtím deaktivováno, je aktivováno.

Argumenty

```
IWatch Interrupt
```

`Interrupt`

Datový typ: `intnum`

Proměnná (identita přerušení) přerušení.

Vykonávání programu

Znovu aktivuje přerušení určeného typu. Přerušení generovaná během času, kdy instrukce `ISleep` byla účinná, jsou ignorována.

Další příklady

Více příkladů instrukce `IWatch` je názorně uvedeno dole.

Příklad 1

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalDI di1,1,sig1int;
  ...
  ISleep sig1int;
  weldpart1;
  IWatch sig1int;
```

Během vykonávání rutiny `weldpart1` nejsou povolena žádná přerušení od signálu `di1`.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_UNKINO`	Číslo přerušení je neznámé. Přerušení, která nebyla přikázána ani zapnuta, nejsou povolena.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.127 IWatch - Aktivuje přerušení

RobotWare - OS

Pokračování

Syntaxe

```
IWatch
  [ Interrupt ':=' ] < variable (VAR) of intnum > ';'
```

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*
Deaktivace přerušení	*ISleep - Deaktivuje přerušení na str 318*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.128 Návěstí - Jméno řádky

Použití

`Label` se používá pro pojmenování řádky v programu. Pomocí instrukce `GOTO` může být toto jméno potom použito k posunutí vykonávání programu v rámci stejné rutiny.

Základní příklady

Následující příklad názorně ukazuje instrukci `Label`:

Příklad 1

```
GOTO next;
...
next:
```

Vykonávání programu pokračuje s instrukcí následující po `next`.

Argumenty

```
Label:
```

`Label`

Identifier

Jméno, které chcete přidělit řádce.

Vykonávání programu

Když provedete tuto instrukci, nic se nestane.

Omezení

Návěstí nesmí být stejné jako

- jakékoliv jiné návěstí v rámci stejné rutiny.
- jakékoliv jiné jméno dat v rámci stejné rutiny.

Návěstí skrývá globální data a rutiny se stejným jménem v rámci rutiny, ve které je umístěno.

Syntaxe

```
<identifier>':'
```

Související informace

Pro informace o	Viz
Identifikátory	*Technická referenční příručka - Přehled RAPID*
Posunutí vykonávání programu k návěstí	*GOTO - Přechází na novou instrukci na str 232*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.129 Load - Načíst programový modul během provádění

Použití

`Load` se používá pro načtení programového modulu do paměti programu během vykonávání.

Načtený programový modul bude přidán k již existujícím modulům v paměti programu.

Programový nebo systémový modul mohou být načteny ve statickém (standard) nebo dynamickém režimu.

Staticky a dynamicky načtené moduly mohou být staženy instrukcí `UnLoad`.

Statický režim

Následující tabulka popisuje, jak různé operace ovlivňují staticky načtené programové nebo systémové moduly.

Typ modulu	Nastavit PP na main z FlexPendantu	Otevřít nový program RAPID
Programový modul	Není ovlivněno	Staženo
Systémový modul	Není ovlivněno	Není ovlivněno

Dynamický režim

Následující tabulka popisuje, jak různé operace ovlivňují dynamicky načtené programové nebo systémové moduly.

Typ modulu	Nastavit PP na main z FlexPendantu	Otevřít nový program RAPID
Programový modul	Staženo	Staženo
Systémový modul	Staženo	Staženo

Základní příklady

Následující příklady názorně ukazují instrukci `Load`:

Viz také *Další příklady na str 329*.

Příklad 1

```
Load \Dynamic, diskhome \File:="PART_A.MOD";
```

Načítá programový modul `PART_A.MOD` z `diskhome` do paměti programu. `diskhome` je předdefinovaná řetězcová konstanta `"HOME:"`. Načtěte programový modul v dynamickém režimu.

Příklad 2

```
Load \Dynamic, diskhome \File:="PART_A.MOD";
Load \Dynamic, diskhome \File:="PART_B.MOD" \CheckRef;
```

Načítá programový modul `PART_A.MOD` do paměti programu, potom je načten `PART_B.MOD`. Jestliže `PART_A.MOD` obsahuje reference na `PART_B.MOD`, `\CheckRef` se může použít ke kontrole nevyřešených referencí pouze když je načten poslední modul. Jestliže `\CheckRef` je použit na `PART_A.MOD`, objeví se chyba řádky a modul nebude načten.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Argumenty

```
Load [\Dynamic] FilePath [\File] [\CheckRef]
```

`[\Dynamic]`

Datový typ: `switch`

Spínač zapíná načítání modulu v dynamickém režimu. Jinak běží načítání ve statickém režimu.

`FilePath`

Datový typ: `string`

Cesta souboru a jméno souboru k souboru, který bude načten do paměti programu. Jméno souboru by mělo být vyřazeno, když se používá argument `\File`.

`[\File]`

Datový typ: `string`

Když je jméno souboru vyřazeno v argumentu `FilePath` , potom musí být definováno s tímto argumentem.

`[\CheckRef]`

Datový typ: `switch`

Po načtení modulu zkontrolujte nevyřešené reference v programové úloze. Jinak nebude žádná kontrola nevyřešených referencí provedena.

Vykonávání programu

Vykonávání programu čeká, až programový modul dokončí načítání, potom pokračuje s další instrukcí.

Nevyřešené reference budou vždy akceptovány operací načítání, jestliže není použit parametr `\CheckRef`, ale bude to chyba při běhu při vykonávání nevyřešené reference.

Po načtení bude programový modul propojen a inicializován. Inicializace načteného modulu nastaví všechny proměnné na úrovni modulu na hodnoty jejich jednotky.

Po jakékoliv chybě z operace načítání včetně nevyřešených referencí při použití přepínače `\CheckRef` nebude už načtený modul k dispozici v paměti programu.

Aby bylo možné získat dobrou programovou strukturu, která je snadná na porozumění i udržování, veškeré načítání a stahování programových modulů by se mělo provádět od hlavního modulu, který je vždy přítomen v paměti programu během vykonávání.

Pro načítání programu, který obsahuje hlavní proceduru k hlavnímu programu (s další hlavní procedurou), viz příklad *Další příklady na str 329* dole.

Další příklady

Více příkladů jak používat instrukci `Load` je názorně uvedeno dole.

Další všeobecné příklady

```
Load \Dynamic, "HOME:/DOORDIR/DOOR1.MOD";
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.129 Load - Načíst programový modul během provádění

RobotWare - OS

Pokračování

Načítá programový modul `DOOR1.MOD` z `HOME:` v adresáři `DOORDIR` do paměti programu. Programový modul je načten v dynamickém režimu.

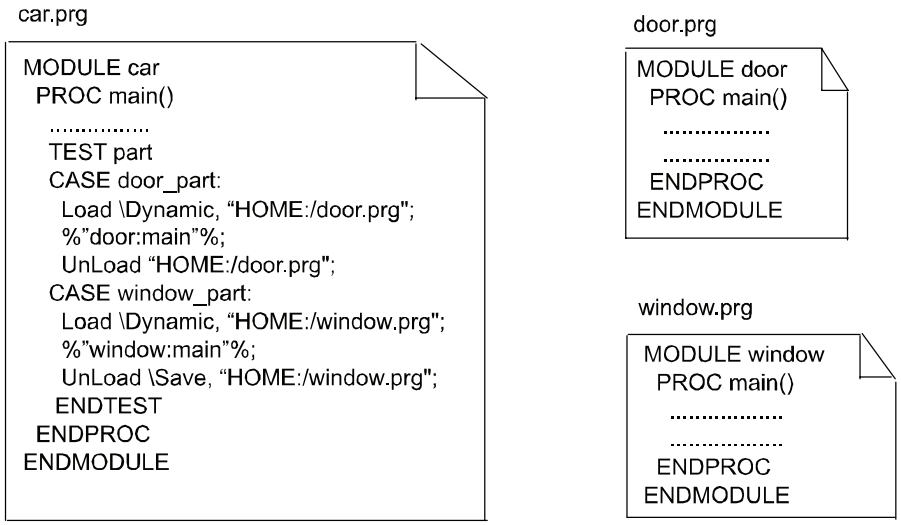
```
Load "HOME:" \File:="DOORDIR/DOOR1.MOD";
```

Stejné jako nahoře, ale jiná syntaxe, a modul je načten ve statickém režimu.

```
Load\Dynamic, "HOME:/DOORDIR/DOOR1.MOD";
%"routine_x"%;
UnLoad "HOME:/DOORDIR/DOOR1.MOD";
```

Procedura `routine_x`, bude provázána během vykonávání (opožděná vazba).

Načtený program obsahuje hlavní proceduru



xx0500002104

Shora uvedený příklad ukazuje, jak můžete načíst program, který obsahuje proceduru `main`. Tento program může být vyvinut a otestován odděleně a později načten s `Load` nebo `StartLoad... WaitLoad` do systému pomocí stejného typu hlavní struktury programu. V tomto příkladu `car.prg`, který načítá jiné programy `door.prg` nebo `window.prg`.

V programu `car.prg` načítáte `door.prg` nebo `window.prg` umístěný na `"HOME:"`. Protože procedury `main` v `door.prg` a `window.prg` jsou po načtení považovány systémem za **LOCAL** v modulu, volání procedury jsou provedena následujícím způsobem: `%"door:main"%` nebo `%"window: main"%`. Tato syntaxe se používá, když chcete získat přístup k procedurám **LOCAL** v jiných modulech, v tomto případě je to procedura `main` v modulu `door` nebo modulu `window`.

Stažení modulů s argumentem `\Save` znovu udělá procedury `main` globálními v uloženém programu.

Nastavíte-li, když je modul `car` nebo `window` načten do systému, ukazatel programu na `main` z jakékoliv části programu, ukazatel programu bude vždy nastaven na globální proceduru `main` v hlavním programu, v tomto příkladu je to `car.prg`.

Omezení

Vyloučit probíhající pohyby robotu během načítání.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_FILNOTFND	Soubor určený v instrukci Load nebyl nalezen.
ERR_IOERROR	Vyskytl se problém se čtením souboru v instrukci Load.
ERR_PRGMEMFULL	Modul nelze načíst, protože paměť programu je zaplněna.
ERR_LOADED	Modul je již načten do paměti programu.
ERR_SYNTAX	Načtený modul obsahuje syntaktické chyby.
ERR_LINKREF	• Výsledkem načteného modulu jsou fatální chyby řádky. • Jestliže se Load používá s přepínačem \CheckRef pro kontrolu referenčních chyb, a paměť programu obsahuje nevyřešené reference.

Jestliže se objevuje některá z těchto chyb, aktuální modul bude stažen a nebude k dispozici v handleru ERROR.

Syntaxe

```
Load
  [´\´Dynamic´,´]
  [FilePath´:=´]<expression (IN) of string>
  [´\´File´:=´ <expression (IN) of string>]
  [´\´CheckRef]´;´
```

Související informace

Pro informace o	Viz
Zrušit načtení programového modulu	*UnLoad - Stáhnout programový modul během provádění na str 905*
Načíst programový modul souběžně s vykonáváním jiného programu	*StartLoad - Načíst programový modul během vykonávání na str 703* *WaitLoad - Připojit načtený modul k úloze na str 947*
Zkontrolovat reference programu	*CheckProgRef - Zkontrolovat reference programu na str 103*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.130 LoadId - Identifikace zatížení nástroje nebo užitečné zátěže

Použití

`LoadId` (*Load Identification*) se může používat pro identifikaci zátěže nástroje (také nástroje chapadla v případě roomfix TCP) nebo užitečné zátěže (aktivuje se s instrukcí `GripLoad`) vykonáním uživatelsky definovaného programu RAPID.

POZNÁMKA

Snazším způsobem, jak identifikovat zatížení nástroje nebo užitečnou zátěž je použít interaktivní RAPID program interaktivního dialogu `LoadIdentify`. Tento program může být spuštěn z nabídky `ProgramEditor/Debug/Call Routine.../LoadIdentify`..

Základní příklady

Následující příklad názorně ukazuje instrukci `LoadId`:

Viz také *Další příklady na str 336*.

Příklad 1

```
VAR bool invalid_pos := TRUE;
VAR jointtarget joints;
VAR bool valid_joints{12};
CONST speeddata low_ori_speed := [20, 5, 20, 5];
VAR bool slow_test_flag := TRUE;
PERS tooldata grip3 := [ TRUE, [[97.4, 0, 223.1], [0.924, 0, 0.383
     ,0]], [0, [0, 0, 0], [1, 0, 0, 0], 0, 0, 0]];
! Check if valid robot type
IF ParIdRobValid(TOOL_LOAD_ID) <> ROB_LOAD_VAL THEN
  EXIT;
ENDIF
! Check if valid robot position
WHILE invalid_pos = TRUE DO
  joints := CJointT();
  IF ParIdPosValid (TOOL_LOAD_ID, joints, valid_joints) = TRUE THEN
    ! Valid position
    invalid_pos := FALSE;
  ELSE
    ! Invalid position
    ! Adjust the position by program movements (horizontal tilt
        house)
    MoveAbsJ joints, low_ori_speed, fine, tool0;
  ENDIF
ENDWHILE
! Do slow test for check of free working area
! Load modules into the system
Load \Dynamic, "RELEASE:/system/mockit.sys";
Load \Dynamic, "RELEASE:/system/mockit1.sys";
IF slow_test_flag = TRUE THEN
  %"LoadId"% TOOL_LOAD_ID, MASS_WITH_AX3, grip3 \SlowTest;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
ENDIF
! Do measurement and update all load data in grip3
%"LoadID"% TOOL_LOAD_ID, MASS_WITH_AX3, grip3;
! Unload modules
UnLoad "RELEASE:/system/mockit.sys";
UnLoad "RELEASE:/system/mockit1.sys";
```

Identifikace zatížení nástroje `grip3`.

Podmínka

Následující podmínky by měly být splněny před měřeními zátěže s `LoadId`:

- Zajistěte, aby všechny zátěže byly správně upevněny na robotu
- Zkontrolujte platný typ robotu s `ParIdRobValid`
- Zkontroluje platnou pozici s `ParIdPosValid`:
  - Osy 3, 5 a 6 se nenacházejí v blízkosti svých pracovních rozsahů
  - Tilthousing téměř vodorovný, tj. osa 4 je v nulové pozici
- Následující data by měla být definována v systémových parametrech a v argumentech k `LoadId` před provedením `LoadId`

  Tabulka dole ukazuje zátěžovou identifikaci nástroje.

Režimy zátěžové identifikace / Definovaná data před `LoadId`	Hmotnost pohybujícího se TCP známá	Hmotnost pohybujícího se TCP neznámá	Roomfix pohybujícího se TCP známý	Roomfix pohybujícího se TCP neznámý
Zátěž horního ramene (Systémový parametr)		Definováno		Definováno
Hmotnost v nástroji	Definováno		Definováno	

  Tabulka dole ukazuje zátěžovou identifikaci užitečné zátěže.

Režimy zátěžové identifikace / Definovaná data před `LoadId`	Hmotnost pohybujícího se TCP známá	Hmotnost pohybujícího se TCP neznámá	Roomfix pohybujícího se TCP známý	Roomfix pohybujícího se TCP neznámý
Zátěž horního ramene (Systémový parametr)		Definováno		Definováno
Zátěžová data v nástroji	Definováno	Definováno	Definováno	Definováno
Hmotnost v užitečné zátěži	Definováno		Definováno	
Rámec nástroje v nástroji	Definováno	Definováno		
Uživatelský rámec v pracovním objektu			Definováno	Definováno
Rámec objektu v pracovním objektu			Definováno	Definováno

- Provozní režim a potlačení rychlosti:
  - Pomalý test se sníženou rychlostí v ručním režimu
  - Měření zátěže v automatickém režimu (nebo ručním režimu s plnou rychlostí) s potlačením rychlosti 100 %

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.130 LoadId - Identifikace zatížení nástroje nebo užitečné zátěže

RobotWare-OS

Pokračování

Argumenty

```
LoadId ParIdType LoadIdType Tool [\PayLoad] [\WObj] [\ConfAngle]
     [\SlowTest] [\Accuracy]
```

`ParIdType`

Datový typ: `paridnum`

Typ identifikace zátěže, jak je definován v tabulce dole.

Hodnota	Symbolická konstanta	Komentář
1	TOOL_LOAD_ID	Identifikovat zatížení nástroje
2	PAY_LOAD_ID	Identifikovat užitečnou zátěž (Ref. instrukce GripLoad)

`LoadIdType`

Datový typ: `loadidnum`

Typ identifikace zátěže, jak je definován v tabulce dole.

Hodnota	Symbolická konstanta	Komentář
1	MASS_KNOWN	Známá hmotnost v nástroji nebo užitečné zátěži. (Hmotnost v určeném Tool nebo PayLoad musí být uvedena)
2	MASS_WITH_AX3	Neznámá hmotnost v nástroji nebo užitečné zátěži. Identifikace hmotnosti v nástroji nebo užitečné zátěži bude provedena s pohyby osy 3

`Tool`

Datový typ: `tooldata`

Proměnná perzistentu pro nástroj, který má být identifikován. Jestliže je určen argument `\PayLoad`, proměnná perzistentu pro nástroj, který se používá.

Pro identifikaci zátěže nástroje by neměly být určovány následující argumenty `\PayLoad` a `\WObj` .

`[ \ PayLoad ]`

Datový typ: `loaddata`

Proměnná perzistentu pro užitečnou zátěž, která má být identifikována.

Tento volitelný argument musí být vždy určen pro zátěžovou identifikaci užitečné zátěže.

`[ \ WObj ]`

Datový typ: `wobjdata`

Proměnná perzistentu pro pracovní objekt, který se používá.

Tento volitelný argument musí být vždy určen pro zátěžovou identifikaci užitečné zátěže s roomfix TCP.

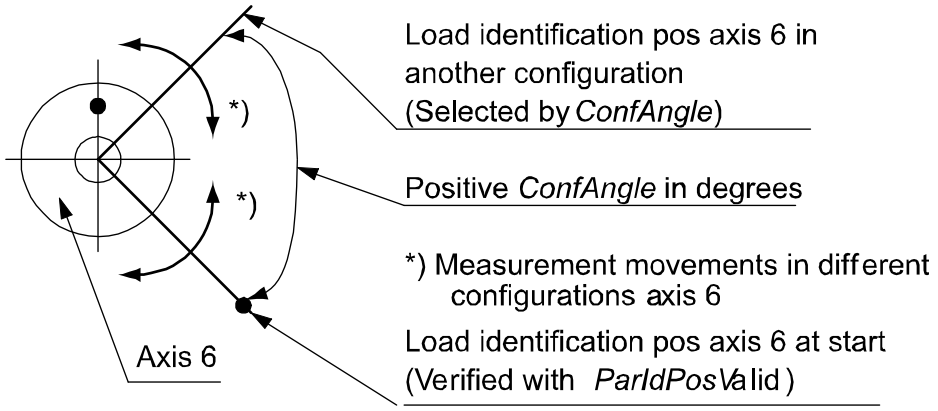
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[ \ ConfAngle ]`

Datový typ: `num`

Volitelný argument pro určení konkrétního úhlu ± stupňů konfigurace, které budou použity pro identifikaci parametru.



xx0500002198

Výchozí hodnota + 90 stupňů, jestliže tento argument není určen. Min + nebo - 30 stupňů. Optimum + nebo - 90 stupňů.

`[ \ SlowTest ]`

Datový typ: `switch`

Volitelný argument pro určení, jestli má být proveden pouze pomalý test pro kontrolu volné pracovní oblasti. Viz tabulka dole:

`LoadId ... \SlowTest`	**Provést pouze pomalý test**
`LoadId ...`	**Provést pouze měření a aktualizovat nástroj nebo užitečnou zátěž**

`[ \ Accuracy ]`

Datový typ: `num`

Proměnná pro výstup vypočítané přesnosti měření v % pro celkový výpočet identifikace zátěže (100 % znamená maximální přesnost).

Vykonávání programu

Robot provede velké množství relativně malých přepravních a měřicích pohybů na osách 5 a 6. Pro identifikaci hmotnosti budou rovněž provedeny pohyby s osou 3.

Po všech měřeních, pohybech a výpočtech zátěže budou data zátěže vrácena do argumentu `Tool` nebo `PayLoad`.. Následující data zátěže jsou vypočítána:

- **Hmotnost v kg (jestliže hmotnost není známa, jinak to není ovlivněno)**
- **Těžiště x, y, z a osy momentu**
- **Setrvačnost ix, iy, iz v kgm**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.130 LoadId - Identifikace zatížení nástroje nebo užitečné zátěže

RobotWare-OS

Pokračování

Další příklady

Více příkladů instrukce `LoadId` je názorně uvedeno dole.

Příklad 1

```
PERS tooldata grip3 := [ FALSE, [[97.4, 0, 223.1], [0.924, 0, 0.383
     ,0]], [6, [10, 10, 100], [0.5, 0.5, 0.5, 0.5], 1.2, 2.7,
     0.5]];
PERS loaddata piece5 := [ 5, [0, 0, 0], [1, 0, 0, 0], 0, 0, 0];
PERS wobjdata wobj2 := [ TRUE, TRUE, "", [ [34, 0, -45], [0.5,
     -0.5, 0.5 ,-0.5] ], [ [0.56, 10, 68], [0.5, 0.5, 0.5 ,0.5] ]
     ];
VAR num load_accuracy;
! Load modules into the system
Load \Dynamic, "RELEASE:/system/mockit.sys";
Load \Dynamic, "RELEASE:/system/mockit1.sys";
! Do measurement and update all payload data except mass in piece5
%"LoadId"% PAY_LOAD_ID, MASS_KNOWN, grip3 \PayLoad:=piece5
     \WObj:=wobj2 \Accuracy:=load_accuracy;
TPWrite " Load accuracy for piece5 (%) = " \Num:=load_accuracy;
! Unload modules
UnLoad "RELEASE:/system/mockit.sys";
UnLoad "RELEASE:/system/mockit1.sys";
```

Zátěžová identifikace užitečné zátěže `piece5` se známou hmotností v instalaci s roomfix TCP.

Omezení

Obvykle se identifikace zátěže nástroje nebo užitečné zátěže pro robot provádí se servisní rutinou `LoadIdentify`. Je také možné provádět tuto identifikaci z instrukcí RAPID `LoadId`. Před načtením nebo vykonáním programu s `LoadId` musí být do systému načteny následující moduly:

```
Load \Dynamic, "RELEASE:/system/mockit.sys";
Load \Dynamic, "RELEASE:/system/mockit1.sys";
```

Potom je možné volat `LoadId` s voláním opožděné vazby (viz příklad 1 nahoře).

Není možné restartovat pohyby identifikace zátěže po jakémkoliv typu zastavení, jako je zastavení programu, nouzové zastavení nebo výpadek elektřiny. Pohyby identifikace zátěže musí být potom spuštěny od začátku.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_PID_MOVESTOP`	U jakékoliv chyby během vykonávání RAPID `NOSTEPIN` rutiny `LoadId`. Ukazatel programu je zvednut k uživatelskému volání `LoadId`.
`ERR_PID_RAISE_PP`	
`ERR_LOADID_FATAL`	

Syntaxe

```
LoadId
  [ ParIdType ':=' ] <expression (IN) of paridnum> ',
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
[ LoadIdType ':=' ] <expression (IN) of loadidnum> ','
[ Tool ':=' ] <persistent (PERS) of tooldata>
[ '\' PayLoad ':=' <persistent (PERS) of loaddata> ]
[ '\' WObj ':=' <persistent (PERS) of wobjdata> ]
[ '\' ConfAngle ':=' <expression (IN) of num> ]
[ '\' SlowTest ]
[ '\' Accuracy ':=' <variable (VAR) of num> ] ';'
```

Související informace

Pro informace o	Viz
Předdefinovaný program Load Identify	*Návod k použití - IRC5 s jednotkou FlexPendant*
Typ identifikace parametru	*paridnum - Typ identifikace parametru na str 1547*
Výsledek `ParIdRobValid`	*paridvalidnum - Výsledek ParIdRobValid na str 1549*
Typ identifikace zátěže	*loadidnum - Identifikace typu zátěže na str 1529*
Platný typ robotu	*ParIdRobValid - Platný typ robotu pro identifikaci parametru na str 1250*
Platná pozice robotu	*ParIdPosValid - Platná pozice robotu pro identifikaci parametru na str 1247*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.131 MakeDir - Vytvořit nový adresář

Použití

`MakeDir` se používá pro vytvoření nového adresáře. Uživatel musí mít právo zápisu a provádění pro základní adresář, pod kterým bude nový adresář vytvořen.

Základní příklady

Následující příklad názorně ukazuje instrukci `MakeDir`:

Příklad 1

```
MakeDir "HOME:/newdir";
```

Tento příklad vytváří nový adresář s názvem `newdir`, pod `HOME`:

Argumenty

```
MakeDir Path
```

`Path`

Datový typ:`string`

Jméno nového adresáře určeného s plnou nebo relativní cestou.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_FILEACC`	Nelze vytvořit adresář.

Syntaxe

```
MakeDir
[ Path':=' ] < expression (IN) of string>';'
```

Související informace

Pro informace o	Viz
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*
Kopírovat soubor	*CopyFile - Kopírovat soubor na str 135*
Zkontrolovat typ souboru	*IsFile - Zkontrolovat typ souboru na str 1207*
Zkontrolujte velikost souboru	*FileSize - Vyhledat velikost souboru na str 1155*
Zkontrolujte velikost souborového systému	*FSSize - Vyhledat velikost souborového systému na str 1161*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.132 ManLoadIdProc - Identifikace zátěže IRBP manipulátorů

Použití

`ManLoadIdProc` (*Manipulator Load Identification Procedure*) se používá pro identifikaci zátěže užitečné zátěže u externích manipulátorů provedením uživatelsky definovaného RAPID programu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

POZNÁMKA

Snazším způsobem, jak identifikovat užitečnou zátěž, je použít servisní rutinu `ManLoadIdentify`. Tato servisní rutina může být spuštěna z nabídky **Editor programu**, **Ladění**, **Volání rutiny**, **ManLoadIdentify**.

Základní příklady

Následující příklady názorně ukazují instrukci `ManLoadIdProc`:

```
PERS loaddata myload := [6,[0,0,0],[1,0,0,0],0,0,0];
VAR bool defined;
ActUnit STN1;
ManLoadIdProc \ParIdType := IRBP_L
  \MechUnit := STN1
  \PayLoad := myload
  \ConfigAngle := 60
  \AlreadyActive
  \DefinedFlag := defined;
DeactUnit STN1;
```

Identifikace zátěže užitečné zátěže `myload` upevněné na mechanické jednotce `STN1`. Externí manipulátor je typu `IRBP-L`. Konfigurační úhel je nastaven na 60 stupňů. Manipulátor je aktivován před identifikací zátěže a deaktivován po ní. Po aktualizaci a definování identifikace `myload` je tato nastavena na `TRUE`.

Argumenty

```
ManLoadIdProc [\ParIdType] [\MechUnit] | [\MechUnitName]
     [\AxisNumber] [\PayLoad] [\ConfigAngle] [\DeactAll] |
     [\AlreadyActive] [DefinedFlag] [DoExit]
```

`[ \ ParIdType ]`

Datový typ: `paridnum`

Typ identifikace parametru. Předdefinované konstanty se nacházejí pod datovým typem `paridnum`.

`[ \ MechUnit ]`

Datový typ: `mecunit`

Mechanická jednotka použitá pro identifikaci zátěže. Nemůže být použita společně s argumentem `\MechUnitName`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.132 ManLoadIdProc - Identifikace zátěže IRBP manipulátorů

RobotWare-OS

Pokračování

`[ \ MechUnitName ]`

Datový typ: `string`

Mechanická jednotka použitá pro identifikaci zátěže, danou jako řetězec. Nemůže být použita společně s argumentem `\MechUnit`.

`[ \ AxisNumber ]`

Datový typ: `num`

Číslo osy v rámci mechanické jednotky, která drží zátěž k identifikaci.

`[ \ PayLoad ]`

Datový typ: `loaddata`

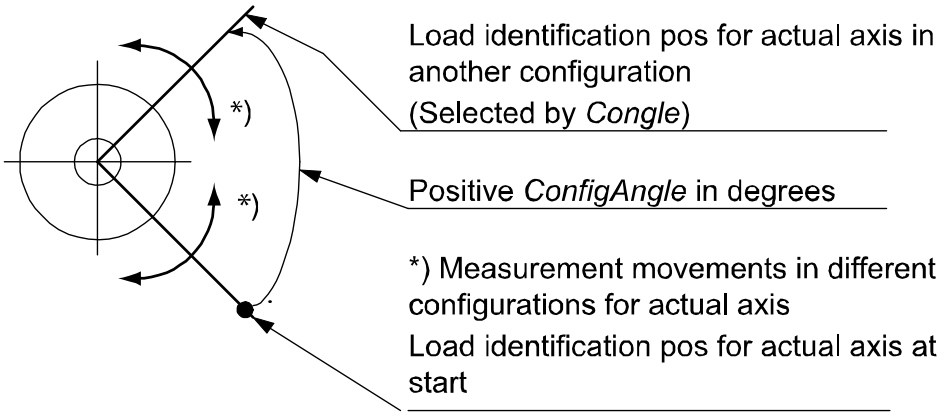
Proměnná pro užitečnou zátěž k identifikaci. Musí být stanovena hmotnost komponentu.

Proměnná bude aktualizována po provedení identifikace.

`[ \ ConfigAngle ]`

Datový typ: `num`

Specifikace konkrétních konfiguračních úhlů ± stupňů, které budou použity pro identifikaci parametru.



xx0500002197

Min. + nebo - 30 stupňů. Optimum + nebo - 90 stupňů.

`[ \ DeactAll ]`

Datový typ: `switch`

Jestliže se používá tento přepínač, všechny mechanické jednotky v systému budou deaktivovány před ukončením identifikace. Mechanická jednotka k identifikaci bude potom aktivována. Nemůže se používat společně s argumentem `\AlreadyActive`.

`[ \ AlreadyActive ]`

Datový typ: `switch`

Tento přepínač se používá, jestliže mechanická jednotka k identifikaci je aktivní. Nemůže se používat společně s argumentem `\DeactAll`.

`[ \ DefinedFlag ]`

Datový typ: `bool`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Tento argument bude nastaven na TRUE, jestliže identifikace byla provedena, jinak FALSE.

[\ DoExit]

Datový typ: bool

Jestliže je nastavena na TRUE, identifikace zátěže skončí s příkazem EXIT, aby přinutila uživatele nastavit PP na main před pokračováním vykonávání. Jestliže není přítomen nebo je nastaven na FALSE, žádný EXIT nebude proveden. Všimněte si, že ManLoadIdProc vždy vyčistí aktuální dráhu.

Vykonávání programu

Všechny argumenty jsou volitelné. Jestliže argument není dán, uživatel bude pořádán o hodnotu z FlexPendantu (kromě \DoExit).

Uživatel bude vždy požádán o zadání hmotnosti, a jestliže manipulátor je typu IRBP R, z v mm.

Mechanická jednotka bude provádět velký počet relativně malých přepravních a měřicích pohybů.

Po všech měřeních, pohybech a výpočtech zátěže budou data zátěže vrácena do argumentu Payload, jestliže se používá. Následující data zátěže jsou vypočítána.

Typ manipulátoru/ Vypočítaná zátěžová data	IRBP-K	IRBP-L IRBP-C IRBP_T	IRBP-R	IRBP-A IRBP-B IRBP-D
Parametr PayLoad - cog.x, cog.y, cog.z v loaddata v mm	cog.x cog.y	cog.x cog.y	cog.x cog.y	cog.x cog.y cog.z
Parametr PayLoad - ix, iy, iz v loaddata v kgm^2	iz	iz	ix iy iz	ix iy iz

Vypočítaná data budou zobrazena na FlexPendantu.

Omezení

Obvykle se zátěžová identifikace zátěže pro externí manipulátor provádí se servisní rutinou ManLoadIdentify. Je také možné provést tuto identifikaci s RAPID instrukcí ManLoadIdProc.

Jakákoliv právě probíhající dráha bude vyčištěna před identifikací zátěže. Ukazatel programu bude ztracen po identifikaci zátěže, jestliže se použije argument \DoExit:=TRUE.

Není možné restartovat pohyby identifikace zátěže po jakémkoliv typu zastavení, jako je zastavení programu, nouzové zastavení nebo výpadek elektřiny. Pohyby identifikace zátěže musí být potom znovu spuštěny od začátku.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.132 ManLoadIdProc - Identifikace zátěže IRBP manipulátorů

RobotWare-OS

Pokračování

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_PID_MOVESTOP	U jakékoliv chyby během vykonávání RAPID NOSTEPIN rutiny ManLoadIdProc. Ukazatel programu je zvednut k uživatelskému volání ManLoadIdProc.
ERR_PID_RAISE_PP	
ERR_LOADID_FATAL	

Syntaxe

```
ManLoadIdProc
  [ '\'ParIdType ':=' <expression (IN) of paridnum>]
  [ '\'MechUnit ':=' <variable (VAR) of mecunit> ]
  | ['\' MechUnitName ':=' <expression (IN) of string>]
  [ '\' AxisNumber ':=' <expression (IN) of num> ]
  [ '\' PayLoad ':=' <var or pers (INOUT) of loaddata>
  [ '\' ConfigAngle ':=' <expression (IN) of num>]
  [ '\' DeactAll] | [ '\' AlreadyActive]
  [ '\' DefinedFlag ':=' <variable (VAR) of bool> ]
  [ '\' DoExit ':=' <expression (IN) of bool> ] ';'
```

Související informace

Pro informace o	Viz
Typ identifikace parametru	*paridnum - Typ identifikace parametru na str 1547*
Mechanická jednotka	*mecunit - Mechanická jednotka na str 1531*
Payload	*loaddata - Zátěžová data na str 1523*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.133 MechUnitLoad - Definuje užitečnou zátěž pro mechanickou jednotku

Použití

`MechUnitLoad` se používá k definování užitečné zátěže pro externí mechanickou jednotku. Užitečná zátěž pro robot je definována s instrukcí `GripLoad`.

Tato instrukce by se měla používat pro všechny mechanické jednotky s dynamickým modelem (ABB polohovadla a trasové pohyby) v servu, aby bylo dosaženo nejlepší pohybové činnosti.

Instrukce `MechUnitLoad` by se měla vždy vykonávat po provedení instrukce `ActUnit`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Popis

`MechUnitLoad` určuje, kterou zátěž nesou mechanické jednotky. Určené zátěže se používají k nastavení dynamického modelu mechanické jednotky tak, aby pohyby mechanické jednotky bylo možné řídit nejlepším možným způsobem.

Užitečná zátěž se připojuje/odpojuje pomocí instrukce `MechUnitLoad`, která přičítá nebo odečítá váhu užitečné zátěže k váze mechanické jednotky.

VAROVÁNÍ

Je důležité vždy definovat skutečnou zátěž mechanické jednotky (tj. upínadlo a pracovní kus). Nesprávné definice zátěžových dat mohou mít za následek přetížení mechanické konstrukce.

Uvedení nesprávných údajů může často vést k následujícím důsledkům:

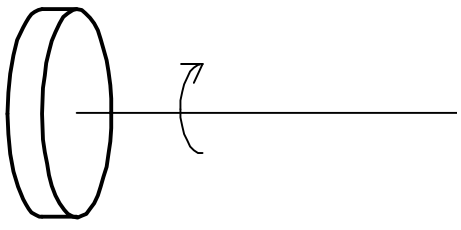
- Mechanická jednotka nebude využita na svoji maximální kapacitu
- Bude narušena přesnost dráhy s rizikem minutí
- Vznikne riziko přetížení mechanické konstrukce

Základní příklady

Následující příklady názorně ukazují instrukci `MechUnitLoad`:

Obrázek

Následující obrázek ukazuje osu 1 na mechanické jednotce pojmenované `STN1` typu IRBP L.



xx0500002142

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.133 MechUnitLoad - Definuje užitečnou zátěž pro mechanickou jednotku

RobotWare - OS

Pokračování

Příklad 1

```
ActUnit SNT1;
MechUnitLoad STN1, 1, load0;
```

Aktivujte mechanickou jednotku `STN1` a definujte užitečnou zátěž `load0` odpovídající (vůbec) žádné zátěži upevněné na ose 1.

Příklad 2

```
ActUnit STN1;
MechUnitLoad STN1, 1, fixture1;
```

Aktivujte mechanickou jednotku `STN1` a definujte užitečnou zátěž `fixture1` odpovídající upínadlu upevněnému na ose 1.

Příklad 3

```
ActUnit STN1;
MechUnitLoad STN1, 1, workpiece1;
```

Aktivujte mechanickou jednotku `STN1` a definujte užitečnou zátěž `workpiece1` odpovídající upínadlu a pracovnímu kusu upevněným na ose 1.

Argumenty

```
MechUnitLoad MechUnit AxisNo Load
```

`MechUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`AxisNo`

Axis Number

Datový typ: `num`

Číslo osy v rámci mechanické jednotky, která drží zátěž. Číslování os začíná od 1.

`Load`

Datový typ: `loaddata`

Zátěžová data, která popisují aktuální užitečnou zátěž, která bude definována, tj. upínadlo nebo upínadlo společně s pracovním kusem podle toho, jestli pracovní kus je upevněn na mechanické jednotce nebo nikoliv.

Vykonávání programu

Po vykonání `MechUnitLoad`, když robot a pomocné osy mají přejít do stavu klidu, je určená zátěž definována pro určenou mechanickou jednotku a osu. To znamená, že užitečná zátěž je řízena a sledována řídicím systémem.

Výchozí užitečná zátěž při používání režimu restartu **Resetovat systém** pro určitý typ mechanické jednotky, je předdefinovaná maximální užitečná zátěž pro tento typ mechanické jednotky.

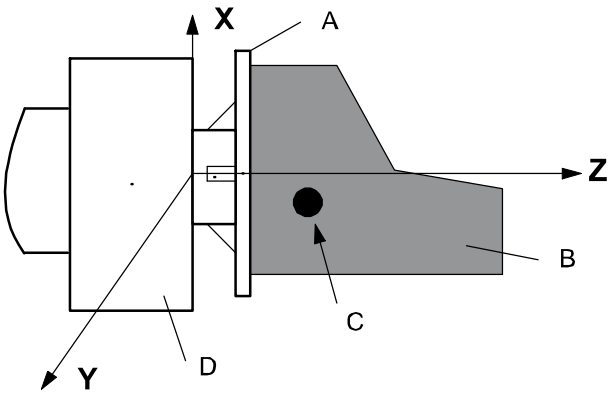
Když se používá další užitečná zátěž, aktuální užitečná zátěž pro mechanickou jednotku a osu by měla být znovu definována s touto instrukcí. To by se mělo provést vždy po aktivaci mechanické jednotky.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Definovaná užitečná zátěž přežije restart. Definovaná užitečná zátěž přežije také restart programu po ruční aktivaci ostatních mechanických jednotek z okna ručního krokového posuvu (jogging).

Následující grafika ukazuje užitečnou zátěž upevněnou na koncovém efektoru mechanické jednotky (souřadný systém koncového efektoru pro mechanickou jednotku).



xx0500002143

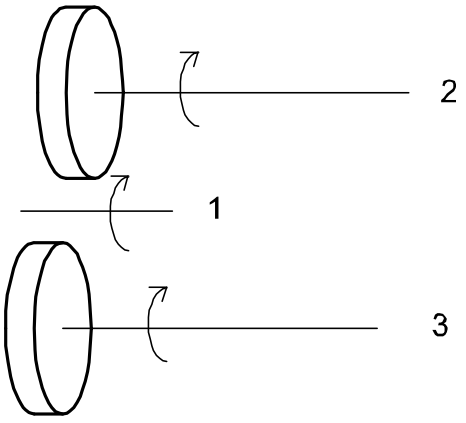
A	Koncový efektor
B	Upínadlo a pracovní kus
C	Těžiště užitečné zátěže (upínadlo + pracovní kus)
D	Mechanická jednotka

Další příklady

Více příkladů jak používat instrukci `MechUnitLoad` je názorně uvedeno dole.

Obrázek

Následující obrázek ukazuje mechanickou jednotku pojmenovanou `INTERCH` typu IRBP K se třemi osami (1, 2, a 3).



xx0500002144

Příklad 1

```
MoveL homeside1, v1000, fine, gun1;
...
ActUnit INTERCH;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.133 MechUnitLoad - Definuje užitečnou zátěž pro mechanickou jednotku

RobotWare - OS

Pokračování

Celá mechanická jednotka `INTERCH` je aktivována.

Příklad 2

```
MechUnitLoad INTERCH, 2, workpiece1;
```

Definuje užitečnou zátěž `workpiece1` na mechanické jednotce `INTERCH`, osa 2.

Příklad 3

```
MechUnitLoad INTERCH, 3, workpiece2;
```

Definuje užitečnou zátěž `workpiece2` na mechanické jednotce `INTERCH`, osa 3.

Příklad 4

```
MoveL homeside2, v1000, fine, gun1;
```

Osy mechanické jednotky `INTERCH` se posunou do pozice spínače `homeside2` s upevněnou užitečnou zátěží na osách 2 a 3.

Příklad 5

```
ActUnit STN1;
MechUnitLoad STN1, 1, workpiece1;
```

Mechanická jednotka `STN1` je aktivována. Definuje užitečnou zátěž `workpiece1` na mechanické jednotce `STN1`, osa 1.

Omezení

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (`zonedata fine`), nikoliv s fly-by bodem, jinak nebude možný restart po selhání napájení.

`MechUnitLoad` se nemůže provádět v RAPID rutině připojené k žádné z následujících speciálních systémových událostí: PowerOn, Stop, QStop, , Restart nebo Step.

Syntaxe

```
MechUnitLoad
  [MechUnit ':=' ] <variable (VAR) of mecunit> ','
  [AxisNo ':=' ] <expression (IN) of num> ','
  [Load ':=' ] <persistent (PERS) of loaddata> ';'
```

Související informace

Pro informace o	Viz
Identifikace užitečné zátěže pro externí mechanické jednotky	*Příručka k produktu - IRBP /D2009*
Definice dat mechanické jednotky	*mecunit - Mechanická jednotka na str 1531*
Definice zátěžových dat	*loaddata - Zátěžová data na str 1523*
Definovat užitečnou zátěž pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.134 MotionProcessModeSet - Nastavit režim pohybového procesu

Použití

`MotionProcessModeSet` se používá pro nastavení režimu pohybového procesu (*Motion Process Mode*) pro TCP robot.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* ve všech pohybových úlohách.

Základní příklady

Následující příklad názorně ukazuje instrukci `MotionProcessModeSet`:

```
MotionProcessModeSet OPTIMAL_CYCLE_TIME_MODE;
! Do cycle-time critical movement
..
MotionProcessModeSet ACCURACY_MODE;
! Do cutting with high accuracy
..
```

Změna režimu pohybového procesu použitého pro TCP robot v době běhu.

Argumenty

```
MotionProcessModeSet Mode
```

`Mode`

Datový typ: `motionprocessmode`

Režim pohybového procesu, který bude použit. Je to celočíselná konstanta datového typu `motionprocessmode`.

Vykonávání programu

Režim pohybového procesu se vztahuje na TCP robot, dokud nebude vykonána nová instrukce `MotionProcessModeSet`, viz *Související informace na str 348*.

Výchozí (default) konfigurovaná hodnota pro režim pohybového procesu je nastavena automaticky:

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.134 MotionProcessModeSet - Nastavit režim pohybového procesu

RobotWare - OS

Pokračování

Předdefinovaná data

Následující symbolické konstanty datového typu `motionprocessmode` jsou předdefinovány a mohou se používat k určení celého čísla v argumentu `Mode`.

Výchozí režim je definován systémovým parametrem *Use Motion Process Mode* v typu *Robot*, téma *Motion*.

Symbolická konstanta	Konstantní hodnota	Popis
`OPTIMAL_CYCLE_TIME_MODE`	1	Tento režim poskytuje nejkratší možný čas cyklu.
`LOW_SPEED_ACCURACY_MODE`	2	Tento režim zvyšuje přesnost dráhy, hlavně u velkých robotů.
`LOW_SPEED_STIFF_MODE`	3	Tento režim se doporučuje u kontaktních aplikací s nízkou rychlostí, kde je důležitá maximální tuhost serva.
`ACCURACY_MODE`	4	Tento režim zvyšuje přesnost dráhy, většinou u malých robotů.
`MPM_USER_MODE_1`	5	Režimy definované uživatelem.
`MPM_USER_MODE_2`	6	
`MPM_USER_MODE_3`	7	
`MPM_USER_MODE_4`	8	

Omezení

Režim je možné měnit pouze když robot stojí v klidu, jinak je vynucen jemný bod.

Syntaxe

```
MotionProcessModeSet
  [ Mode ':=' ] < expression (IN) of motionprocessmode> ';'
```

Související informace

Pro informace o	Viz
Advanced robot motion	*Application manual - Controller software IRC5*
Konfigurace parametrů *Motion Process Mode*.	*Technická referenční příručka - Systémové parametry*
Ladění serv.	*TuneServo - Ladění serv na str 886*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.135 MotionSup - Deaktivuje/aktivuje dohled (supervizi) pohybu

Použití

`MotionSup` (*Motion Supervision*) se používá pro deaktivaci nebo aktivaci funkce dohledu (supervize) nad pohyby robotu během vykonávání programu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Popis

Dohled (supervize) pohybu je název sady funkcí pro vysoce citlivý dohled nad robotem podle jeho modelu. Obsahuje funkci dohledu zatížení, dohledu zaseknutí a detekci kolizí. Jelikož dohled je navržen jako velmi citlivý, může se přerušit, jestliže vzniknou velké procesní síly, které působí na robot.

Jestliže zatížení není správně definováno, použijte pro jeho stanovení funkci identifikace zátěže. Jestliže jsou ve většině částí aplikace přítomny velké externí procesní síly, jako během začišťování, potom použijte systémové parametry na zvýšení úrovně dohledu, dokud přerušování nezmizí. Jestliže se externí síly vyskytují jen občasně, jako během zavírání velké pistole pro bodové svařování, potom by měla být použita instrukce `MotionSup` pro zvýšení úrovně dohledu (nebo pro vypnutí funkce) pro ty části aplikace, kde se poruchy objevují.

Základní příklady

Následující příklad názorně ukazuje instrukci `MotionSup`:

Příklad 1

```
! If the motion supervision is active in the system parameters,
! then it is active by default during program execution
...
! If the motion supervision is deactivated through the system
! parameters,
! then it cannot be activated through the MotionSup instruction
...
! Deactivate motion supervision during program execution
MotionSup \Off;
...
! Activate motion supervision again during program execution
MotionSup \On;
...
! Tune the supervision level to 200% (makes the function less
! sensitive) of the level in
! the system parameters
MotionSup \On \TuneValue:= 200;
...
```

Argumenty

```
MotionSup[\On] | [\Off] [\TuneValue]
```

[\On]

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.135 MotionSup - Deaktivuje/aktivuje dohled (supervizi) pohybu

Collision Detection

Pokračování

Aktivovat funkci dohledu pohybu během vykonávání programu (jestliže to bylo již aktivováno v systémových parametrech).

[\Off]

Datový typ: switch

Deaktivovat funkci dohledu pohybu během vykonávání programu.

Musí být určen jeden z argumentů \On nebo \Off.

[\TuneValue]

Datový typ: num

Ladění úrovně citlivosti dohledu pohybu v procentech (1 - 300 %) úrovně systémových parametrů. Vyšší úroveň dává robustnější citlivost. Tento argument může být kombinován pouze s argumentem \On.

Vykonávání programu

Jestliže je funkce dohledu pohybu aktivní v systémových parametrech i v programu RAPID a dohled pohybu se spustí kvůli kolizi atd., potom

- robot se zastaví tak rychle, jak je to možné
- robot se vrátí, aby odstranil všechny zbytkové síly
- vykonávání programu se zastaví s chybovou zprávou

Jestliže dohled pohybu je aktivní v systémových parametrech, je potom aktivní standardně během vykonávání programu (TuneValue 100 %). Tyto hodnoty jsou nastaveny automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k main
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Omezení

Dohled pohybu není nikdy aktivní u externích os nebo když jeden nebo více spojů běží v nezávislém režimu spojů. Při používání robotu s režimu měkkého serva může být nutné vypnout dohled pohybu, aby nedocházelo k náhodnému přerušování.

Syntaxe

```
MotionSup
[ ’\’ On] | [ ’\’ Off ]
[’\’ Tunevalue’:=’< expression (IN) of num> ] ’;´
```

Související informace

Pro informace o	Viz
Všeobecný popis funkce	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Ladění pomocí systémových parametrů	*Technická referenční příručka - Systémové parametry*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.136 MoveAbsJ - Posune robot do absolutní pozice spoje

Použití

`MoveAbsJ` (*Move Absolute Joint*) se používá pro posun robotu a externích os do absolutní pozice definované v pozicích os.

Příklady použití

- koncový bod je singulárním bodem
- pro víceznačné pozice na IRB 6400C, např. pro pohyby s nástrojem přes robot

Konečná pozice robotu během pohybu s `MoveAbsJ` není ovlivněna ani daným nástrojem a pracovním objektem, ani aktivním posunem programu. Robot používá tato data k výpočtu zatížení, rychlosti TCP a rohové dráhy. Stejné nástroje je možné použít v sousedních pohybových instrukcích.

Robot a externí osy se pohybují k cílové pozici podél nelineární dráhy. Všechny osy dosáhnout své cílové pozice současně.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `MoveAbsJ`:

Viz také *Další příklady na str 355*.

Příklad 1

```
MoveAbsJ p50, v1000, z50, tool2;
```

Robot s nástrojem `tool2` se pohybuje podél nelineární dráhy k absolutní pozici osy, `p50`, s daty rychlosti `v1000` a daty zóny `z50`.

Příklad 2

```
MoveAbsJ *, v1000\T:=5, fine, grip3;
```

Robot s nástrojem `grip3` se pohybuje podél nelineární dráhy ke stop bodu, který je uložen jako absolutní pozice osy v instrukci (označeno s *). Celý pohyb trvá `5` sekund.

Argumenty

```
MoveAbsJ [\Conc] ToJointPos [\ID] [\NoEOffs] Speed [\V] | [\T] Zone
     [\Z] [\Inpos] Tool [\WObj] [\TLoad]
```

`[\Conc]`

Concurrent

Datový typ:`switch`

Následné instrukce jsou vykonávány, zatímco robot se pohybuje. Argument se obvykle nepoužívá, ale je použit ke zkrácení doby cyklu, když, například, komunikuje s externím vybavením, jestliže synchronizace není žádoucí.

Použitím argumentu `\Conc` je počet pohybových instrukcí za sebou omezen na 5. V programové sekci, která obsahuje `StorePath-RestoPath`, nejsou pohybové instrukce s argumentem `\Conc` povoleny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže tento argument je vypuštěn a `ToJointPos` není stop bodem, následná instrukce je vykonána předtím, než robot dosáhne naprogramované zóny.

Tento argument se nemůže používat v koordinovaném synchronizovaném pohybu v systému MultiMove.

`ToJointPos`

To Joint Position

Datový typ: `jointtarget`

Cílová absolutní pozice spoje robotu a externích os. Je definována jako jmenovitá pozice nebo uložena přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`[ \NoEOffs ]`

No External Offsets

Datový typ: `switch`

Jestliže je nastaven argument `\NoEOffs`, potom pohyb s `MoveAbsJ` není ovlivněn aktivními ofsety pro externí osy.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \V ]`

Velocity

Datový typ: `num`

Tento argument se používá k určení rychlosti TCP v mm/sek. přímo v instrukci. Je potom vyměněn za odpovídající rychlost, určenou v rychlostních datech.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ:`zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.136 MoveAbsJ - Posune robot do absolutní pozice spoje
RobotWare - OS
Pokračování

`[ \Z ]`

`Zone`

Datový typ: `num`

Tento argument se používá k určení přesnosti pozice TCP robotu přímo v instrukci. Délka rohové dráhy je dána v mm, což je vyměněno za odpovídající zónu, která je určena v datech zóny.

`[ \Inpos ]`

In position

Datový typ: `stoppointdata`

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

`Tool`

Datový typ: `tooldata`

Nástroj používaný během pohybu.

Pozice TCP a zátěž na nástroji jsou definovány jako data nástroje. Pozice TCP se používá pro výpočet rychlosti a rohové dráhy pro pohyb.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt použitý během pohybu.

Tento argument může být vypuštěn, jestliže nástroj je držen robotem. Jestliže robot drží pracovní objekt, tj. nástroj je stacionární nebo s koordinovanými externími osami, potom musí být argument stanoven.

V případě stacionárního nástroje nebo koordinovaných externích os jsou data použitá v systému pro výpočet rychlosti a rohová dráha pro pohyb definovány v pracovním objektu.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

Pohyb s `MoveAbsJ` není ovlivněn posunem aktivního programu, a jestliže je vykonáván s přepínačem `\NoEOffs`, nebude tam žádný ofset pro externí osy. Bez přepínače `\NoEOffs` jsou externí osy v cíli určení ovlivněny aktivním ofsetem pro externí osy.

Nástroj je veden do absolutní pozice určení spoje s interpolací úhlů osy. To znamená, že každá osa je vedena konstantní rychlostí osy a že všechny osy dosáhnou pozice určení spoje současně, což bude mít za následek nelineární dráhu.

Obecně řečeno, TCP je veden přibližnou naprogramovanou rychlostí, Nástroj je reorientován a externí osy jsou v pohybu ve stejnou dobu, kdy se pohybuje TCP. Jestliže naprogramované rychlosti pro reorientaci nebo pro externí osy nemůže být dosaženo, rychlost TCP bude snížena.

Rohová dráha je obvykle generována, když je přenášen pohyb k další sekci dráhy. Jestliže stop bod je určen v datech zóny, vykonávání programu pokračuje pouze v případě, že robot a externí osy dosáhly příslušné pozice spoje.

Další příklady

Více příkladů jak používat instrukci `MoveAbsJ` je názorně uvedeno dole.

Příklad 1

```
MoveAbsJ *, v2000\V:=2200, z40 \Z:=45, grip3;
```

Nástroj `grip3` je v pohybu podél nelineární dráhy k absolutní pozici spoje uložené v instrukci. Pohyb je prováděn s daty nastavenými na `v2000` a `z40`. Rychlost a velikost zóny TCP jsou `2200` mm/sek. a `45` mm.

Příklad 2

```
MoveAbsJ p5, v2000, fine \Inpos := inpos50, grip3;
```

Nástroj `grip3` je v pohybu podél nelineární dráhy k absolutní pozici spoje `p5`. Robot to považuje za bod, kde je splněno 50 % poziční podmínky a 50 % rychlostní podmínky pro stop bod `fine`. Čeká nejdéle 2 sekundy na splnění podmínek. Viz předdefinovaná data `inpos50` datového typu `stoppointdata`.

Příklad 3

```
MoveAbsJ \Conc, *, v2000, z40, grip3;
```

Nástroj `grip3` je v pohybu podél nelineární dráhy k absolutní pozici spoje uložené v instrukci. Následné logické instrukce jsou vykonány za pohybu robotu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.136 MoveAbsJ - Posune robot do absolutní pozice spoje

RobotWare - OS

Pokračování

Příklad 4

```
MoveAbsJ \Conc, * \NoEOffs, v2000, z40, grip3;
```

Stejný pohyb jako nahoře, ale pohyb není ovlivněn aktivními ofsety pro externí osy.

Příklad 5

```
GripLoad obj_mass;
MoveAbsJ start, v2000, z40, grip3 \WObj:= obj;
```

Robot posunuje pracovní objekt `obj` ve vztahu k pevnému nástroji `grip3` podél nelineární dráhy k absolutní pozici osy `start`.

Řešení chyb

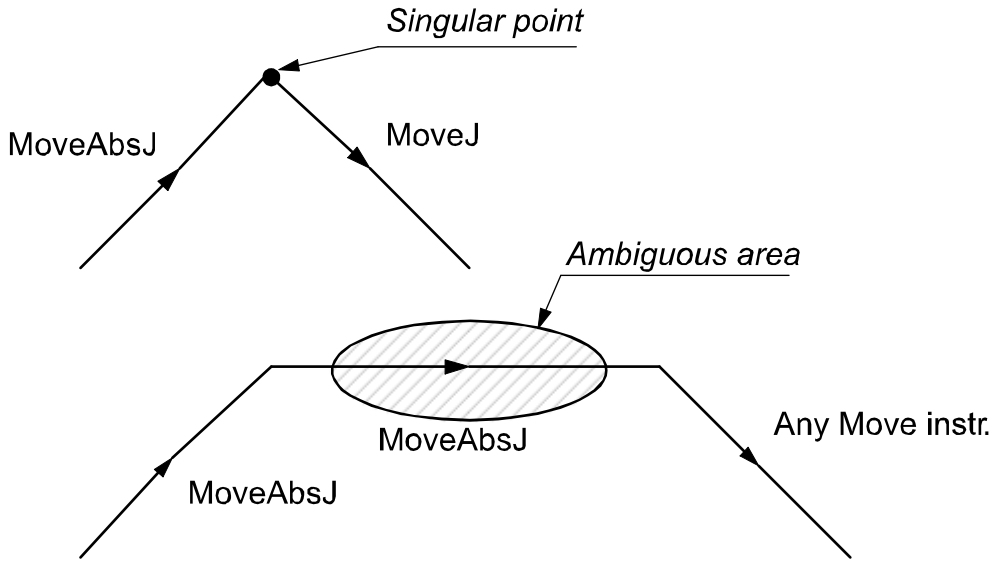
Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_CONC_MAX`	Počet pohybových instrukcí za sebou pomocí argumentu `\Conc` byl překročen.

Omezení

Aby byl možný zpětný běh se zapojenou instrukcí `MoveAbsJ` vyhýbající se problémům se singulárními body nebo nejednoznačnými oblastmi, je zásadní, aby následné instrukce splňovaly určité požadavky, jak následují (viz obrázek dole).

Obrázek ukazuje omezení pro zpětné vykonávání s `MoveAbsJ`.



xx0500002201

Syntaxe

```
MoveAbsJ
  [ '\' Conc ',' ]
  [ ToJointPos ':=' ] < expression (IN) of jointtarget >
  [ '\' ID ':=' < expression (IN) of identno >]
  [ '\' NoEoffs ] ','
  [ Speed ':=' ] < expression (IN) of speeddata >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
[ '\' V ':=' < expression (IN) of num > ]
| [ '\' T ':=' < expression (IN) of num > ] ','
[Zone ':=' ] < expression (IN) of zonedata>
['\' Z ':=' ] < expression (IN) of num >
[ '\' Inpos' :=' < expression (IN) of stoppointdata > ] ','
[ Tool ':=' ] < persistent (PERS) of tooldata >
[ '\' WObj ':=' < persistent (PERS) of wobjdata > ]
[ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Definice jointtarget	*jointtarget - Data pozice svaru na str 1520*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice dat stop bodu	*stoppointdata - Data stop bodu na str 1590*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souběžné vykonávání programu	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.137 MoveC - Pohybuje robotem kruhově

Použití

`MoveC` se používá k posunu středního bodu nástroje (TCP) kruhově na dané místo určení. Během pohybu zůstává orientace normálně nezměněna ve vztahu ke kruhu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `MoveC`:

Viz také *Další příklady na str 362*.

Příklad 1

```
MoveC p1, p2, v500, z30, tool2;
```

TCP nástroje `tool2`, je posunut kruhově na pozici `p2` s rychlostními daty `v500` a zónovými daty `z30`. Kruh je definován od pozice startu, bodu kruhu `p1` a cílového bodu `p2`.

Příklad 2

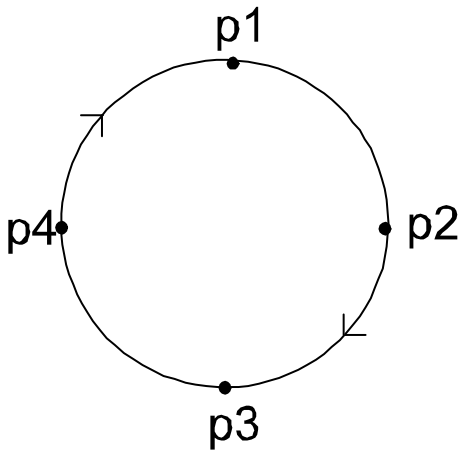
```
MoveC *, *, v500 \T:=5, fine, grip3;
```

TCP nástroje `grip3` je posunut kruhově k jemnému bodu uloženému v instrukci (označeno druhou *). Bod kruhu je také uložen v instrukci (označeno první *). Kompletní pohyb trvá `5` sekund.

Příklad 3

```
MoveL p1, v500, fine, tool1;
MoveC p2, p3, v500, z20, tool1;
MoveC p4, p1, v500, fine, tool1;
```

Obrázek ukazuje, jak je kompletní kruh proveden dvěma instrukcemi `MoveC` .



xx0500002212

Argumenty

```
MoveC [\Conc] CirPoint ToPoint [\ID] Speed [\V] | [\T] Zone [\Z]
     [\Inpos] Tool [\WObj] [\Corr] [\TLoad]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[ \Conc ]`

Concurrent

Datový typ:`switch`

Následné instrukce jsou vykonány během pohybu robotu. Argument se obvykle nepoužívá, ale může se použít kvůli zabránění nechtěným zastavením způsobeným přetíženým CPU při používání fly-by bodů. To je vhodné, když naprogramované body jsou velmi blízko sebe při velkých rychlostech. Argument je také užitečný, když například komunikace s externím vybavením a synchronizace mezi externím vybavením a pohybem robotu nejsou žádoucí.

Použitím argumentu `\Conc` je počet pohybových instrukcí za sebou omezen na 5. V programové sekci, která obsahuje `StorePath-RestoPath`, nejsou pohybové instrukce s argumentem `\Conc` povoleny.

Jestliže tento argument je vypuštěn a `ToPoint` není stop bodem, následná instrukce je vykonána předtím, než robot dosáhne naprogramované zóny.

Tento argument se nemůže používat v koordinovaném synchronizovaném pohybu v systému MultiMove.

`CirPoint`

Datový typ:`robtarget`

Kruhový bod robotu. Kruhový bod je pozice na kruhu mezi start bodem a bodem určení. Aby bylo dosaženo největší přesnosti, měl by být umístěn asi na polovině vzdálenosti mezi body startu a určení. Jestliže je umístěn příliš blízko bodu startu nebo blízko koncového bodu, robot může vydat varování. Střední bod je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci). Pozice externích os se nepoužívají.

`ToPoint`

Datový typ:`robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \V ]`

Velocity

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.137 MoveC - Pohybuje robotem kruhově
RobotWare - OS
Pokračování

Datový typ: num

Tento argument se používá k určení rychlosti TCP v mm/sek. přímo v instrukci. Je potom vyměněn za odpovídající rychlost, určenou v rychlostních datech.

[\T]

Time

Datový typ: num

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se pohybuje robot a externí osy. Je potom vyměněn za odpovídající rychlostní data.

Zone

Datový typ:zonedata

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

[\Z]

Zone

Datový typ: num

Tento argument se používá k určení přesnosti pozice TCP robotu přímo v instrukci. Délka rohové dráhy je dána v mm, což je vyměněno za odpovídající zónu, která je určena v datech zóny.

[\Inpos]

In position

Datový typ:stoppoint data

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru Zone.

Tool

Datový typ: tooldata

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného pozice.

[\WObj]

Work Object

Datový typ: wobjdata

Pracovní objekt (souřadný systém objektu), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnicovému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být stanoven kvůli kruhu ve vztahu k pracovnímu objektu, který bude proveden.

[\Corr]

Correction

Datový typ: switch

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Korekční data zapsaná do vstupu korekcí instrukcí `CorrWrite` budou přidána k pozici dráhy a destinace, jestliže tento argument je přítomen.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

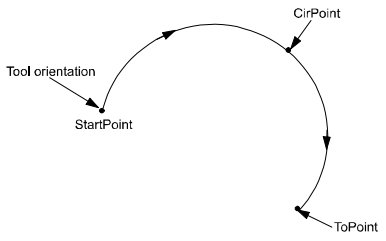
Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

Robot a externí jednotky jsou posunovány k bodu destinace následovně:

- TCP nástroje je posunováno kruhově konstantní naprogramovanou rychlostí.
- Nástroj je reorientován konstantní rychlostí od orientace na startovní pozici k orientaci u bodu určení.
- Reorientace se provádí ve vztahu ke kruhové dráze. Tedy, jestliže orientace ve vztahu k dráze je stejná na bodech startu a určení, vztažná orientace zůstává nezměněna během pohybu (viz obrázek dole).

Obrázek ukazuje orientaci nástroje během kruhového pohybu.



xx0500002214

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.137 MoveC - Pohybuje robotem kruhově
RobotWare - OS
Pokračování

Orientace v bodu kruhu není dosaženo. Je pouze použita k rozlišení mezi dvěma možnými směry reorientace. Přesnost reorientace podél dráhy závisí pouze na orientaci v bodech startu a určení.

Různé režimy pro orientaci nástroje během kruhové dráhy jsou popsány v instrukci `CirPathMode`.

Nekoordinované externí osy jsou vykonávány konstantní rychlostí tak, aby dorazily do bodu určení současně s osami robotu. Pozice v pozici kruhu se nepoužívá.

Jestliže není možné dosáhnout naprogramované rychlosti pro reorientaci nebo pro externí osy, rychlost TCP bude snížena.

Rohová dráha je obvykle generována, když je přenášen pohyb k další sekci dráhy. Jestliže stop bod je určen v datech zóny, vykonávání programu pokračuje pouze v případě, že robot a externí osy dosáhly příslušné pozice.

Další příklady

Více příkladů jak používat instrukci `MoveC` je názorně uvedeno dole.

Příklad 1

```
MoveC *, *, v500 \V:=550, z40 \Z:=45, grip3;
```

TCP nástroje `grip3` je v pohybu kruhově k pozici uložené v instrukci. Pohyb je prováděn s daty nastavenými na `v500` a `z40`. Rychlost a velikost zóny TCP jsou `550` mm/sek. a `45` mm.

Příklad 2

```
MoveC p5, p6, v2000, fine \Inpos := inpos50, grip3;
```

TCP nástroje `grip3` je v pohybu kruhově ke stop bodu `p6`. Robot to považuje za bod, kde je splněno 50 % poziční podmínky a 50 % rychlostní podmínky pro stop bod `fine`. Čeká nejdéle 2 sekundy na splnění podmínek. Viz předdefinovaná data `inpos50` datového typu `stoppointdata`..

Příklad 3

```
MoveC \Conc, *, *, v500, z40, grip3;
```

TCP nástroje `grip3` se pohybuje kruhově k pozici uložené v instrukci. Bod kruhu je také uložen v instrukci. Následné logické instrukce jsou vykonávány za pohybu robotu.

Příklad 4

```
MoveC cir1, p15, v500, z40, grip3 \WObj:=fixture;
```

TCP nástroje `grip3` se pohybuje kruhově k pozici `p15` přes bod kruhu `cir1`. Tyto pozice jsou určeny v souřadném systému objektu pro `fixture`.

Řešení chyb

Jestliže počet pohybových instrukcí za sebou pomocí argumentu `\Conc` byl překročen, potom je systémová proměnná `ERRNO` nastavena na `ERR_CONC_MAX`.

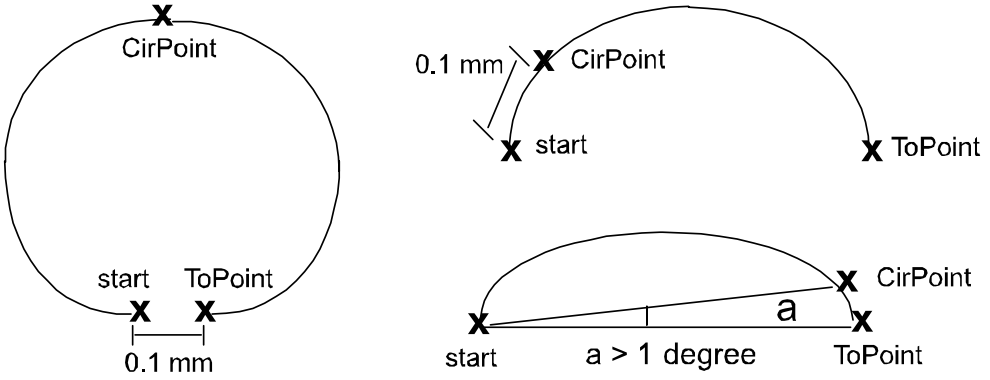
Tato chyba může být zpracována v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Existují omezení v tom, jak může být umístěno `CirPoint` a `ToPoint`, jak ukazuje obrázek dole.



xx0500002213

- Min. vzdálenost mezi startem a `ToPoint` je 0,1 mm.
- Min. vzdálenost mezi startem a `CirPoint` je 0,1 mm.
- Min. úhel mezi `CirPoint` a `ToPoint` od bodu startu je 1 stupeň.

Přesnost může být špatná blízko limitů, například, jestliže bod startu a `ToPoint` na kružnici jsou blízko sebe, porucha způsobená sklonem kruhu může být mnohem větší než přesnost, se kterou byly body programovány.

Zajistěte, aby robot mohl dosáhnout kruhového bodu během provádění programu a podle potřeby rozdělte pořadí kruhového pohybu.

Změna režimu provádění od dopředu na dozadu nebo opačně, zatímco robot je zastaven na kruhové dráze, není přípustná a výsledkem bude chybová zpráva.

VAROVÁNÍ

Instrukce `MoveC` (nebo jiná instrukce obsahující kruhový pohyb) by nikdy neměla být spouštěna od začátku, s TCP mezi bodem kruhu a koncovým bodem. Jinak robot nevezme naprogramovanou dráhu (polohování kolem kruhové dráhy v jiném směru v porovnání s naprogramovanou).

Kvůli minimalizaci rizika nastavte systémový parametr *Restrict placing of circlepoints* na TRUE (typ *Motion Planner*, téma *Motion*).. Parametr přidává dohled, že kruhová dráha se netočí kolem o více než 240 stupňů a že bod kruhu je umístěn ve střední části kruhové dráhy.

Syntaxe

```
MoveC
  [ '\' Conc ',' ]
  [ CirPoint ':=' ] < expression (IN) of robtarget> ','
  [ ToPoint ':=' ] < expression (IN) of robtarget> ','
  [ '\' ID ':=' < expression (IN) of identno>] ','
  [ Speed ':=' ] < expression (IN) of speeddata>
  [ '\' V ':=' < expression (IN) of num> ]
  | [ '\' T ':=' < expression (IN) of num> ] ','
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.137 MoveC - Pohybuje robotem kruhově

RobotWare - OS

Pokračování

```
[ Zone ':=' ] < expression (IN) of zonedata>
[ '\' Z ':=' < expression (IN) of num> ]
[ '\' Inpos ':=' < expression (IN) of stoppointdata> ] ','
[ Tool ':=' ] < persistent (PERS) of tooldata>
[ '\' WObj ':=' < persistent (PERS) of wobjdata> ]
[ '\' Corr ]
[ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice dat stop bodu	*stoppointdata - Data stop bodu na str 1590*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Zapisuje do vstupu korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
Reorientace nástroje během kruhové dráhy	*CirPathMode - Reorientace nástroje během kruhové dráhy na str 105*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Souběžné vykonávání programu	*Technická referenční příručka - Přehled RAPID*
Systémové parametry	*Technická referenční příručka - Systémové parametry*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.138 MoveCAO - Posouvá robot kruhově a nastavuje analogový výstup v rohu

Použití

`MoveCAO` (*Move Circular Analog Output*) se používá pro pohyb středového bodu nástroje (TCP) kruhově k dané destinaci. Určený analogový výstup se nastavuje do středu rohové dráhy u bodu určení. Během pohybu zůstává orientace normálně nezměněna ve vztahu ke kruhu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `MoveCAO`:

Příklad 1

```
MoveCAO p1, p2, v500, z30, tool2, ao1, 1.1;
```

TCP nástroje `tool2`, je posunut kruhově na pozici `p2` s rychlostními daty `v500` a zónovými daty `z30`. Kruh je definován od pozice startu, bodu kruhu `p1` a cílového bodu `p2`. Výstup `ao1` je nastaven ve středu rohové dráhy u `p2`.

Argumenty

```
MoveCAO CirPoint ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal
    Value [\TLoad]
```

`CirPoint`

Datový typ: `robtarget`

Kruhový bod robotu. Kruhový bod je pozice na kruhu mezi start bodem a bodem určení. Aby bylo dosaženo největší přesnosti, měl by být umístěn asi na polovině vzdálenosti mezi body startu a určení. Jestliže je umístěn příliš blízko bodu startu nebo blízko bodu určení, robot může vydat varování. Bod kruhu je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci). Pozice externích os se nepoužívají.

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.138 MoveCAO - Posouvá robot kruhově a nastavuje analogový výstup v rohu
RobotWare - OS
Pokračování

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se pohybuje robot a externí osy. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného pozice.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadný systém objektu), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven kvůli kruhu ve vztahu k pracovnímu objektu, který bude proveden.

`Signal`

Datový typ: `signalao`

Jméno analogového výstupního signálu, který bude změněn.

`Value`

Datový typ: `num`

Požadovaná hodnota signálu.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užitečné zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

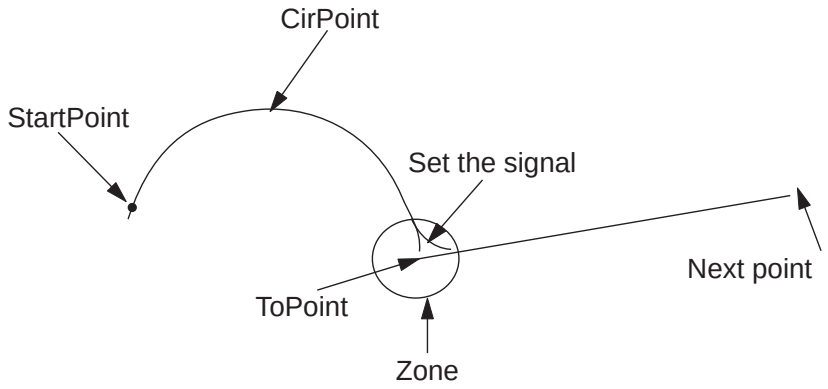
Vykonávání programu

V instrukci `MoveC` najdete více informací o kruhovém pohybu, *MoveC - Pohybuje robotem kruhově na str 358*.

Analogový výstupní signál se nastavuje ve středu rohové dráhy pro flying body, jak je vidět na obrázku dole.

Obrázek ukazuje nastavení analogového výstupního signálu v rohové dráze s `MoveCAO`.

```
MoveCAO p2, p2, v500, z30, tool2, ao1, 1.1;
```



xx1400001116

U stop bodů doporučujeme použít „normální“ programovací sekvenci s `MoveC` a `SetAO`. Ale při používání stop bodu v instrukci `MoveCAO` se analogový výstupní signál nastavuje, když robot dosáhne stop bodu.

Určený I/O signál se nastavuje v režimu vykonávání nepřerušovaně a krokově dopředu, ale nikoliv krokově dozadu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.138 MoveCAO - Posouvá robot kruhově a nastavuje analogový výstup v rohu

RobotWare - OS

Pokračování

Omezení

Všeobecná omezení podle instrukce `MoveC`, viz *MoveC - Pohybuje robotem kruhově na str 358*.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_AO_LIM`, jestliže naprogramovaný argument `Value` pro určený analogový vstupní signál `Signal` je mimo limity.

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
MoveCAO
  [ CirPoint ':=' ] < expression (IN) of robtarget > ','
  [ ToPoint ':=' ] < expression (IN) of robtarget > ','
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
  [ Signal ':=' ] < variable (VAR) of signalao > ] ','
  [ Value ':=' ] < expression (IN) of num > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Posunout robot kruhově	*MoveC - Pohybuje robotem kruhově na str 358*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Pohyby s I/O nastaveními	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O System, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.139 MoveCDO - Posouvá robot kruhově a nastavuje digitální výstup v rohu

Použití

`MoveCDO` (*Move Circular Digital Output*) se používá pro pohyb středového bodu nástroje (TCP) kruhově k dané destinaci. Určený digitální výstup se nastavuje/resetuje do středu rohové dráhy u bodu určení. Během pohybu zůstává orientace normálně nezměněna ve vztahu ke kruhu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `MoveCDO`:

Příklad 1

```
MoveCDO p1, p2, v500, z30, tool2, do1,1;
```

TCP nástroje `tool2`, je posunut kruhově na pozici `p2` s rychlostními daty `v500` a zónovými daty `z30`. Kruh je definován od pozice startu, bodu kruhu `p1` a cílového bodu `p2`. Výstup `do1` je nastaven ve středu rohové dráhy u `p2`.

Argumenty

```
MoveCDO CirPoint ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal
    Value [\TLoad]
```

`CirPoint`

Datový typ: `robtarget`

Kruhový bod robotu. Kruhový bod je pozice na kruhu mezi start bodem a bodem určení. Aby bylo dosaženo největší přesnosti, měl by být umístěn asi na polovině vzdálenosti mezi body startu a určení. Jestliže je umístěn příliš blízko bodu startu nebo blízko bodu určení, robot může vydat varování. Bod kruhu je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci). Pozice externích os se nepoužívají.

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se pohybuje robot a externí osy. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného pozice.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadný systém objektu), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven kvůli kruhu ve vztahu k pracovnímu objektu, který bude proveden.

`Signal`

Datový typ: `signaldo`

Jméno digitálního výstupního signálu, který bude změněn.

`Value`

Datový typ: `dionum`

Požadovaná hodnota signálu (0 nebo 1).

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.139 MoveCDO - Posouvá robot kruhově a nastavuje digitální výstup v rohu
RobotWare - OS
Pokračování

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užitečné zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

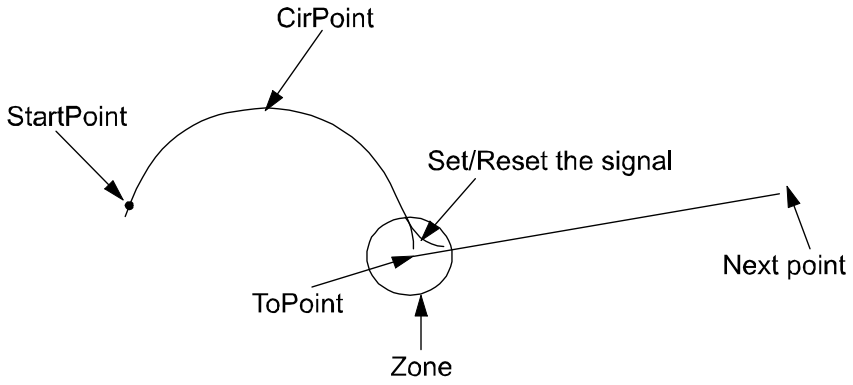
Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

V instrukci `MoveC` najdete více informací o kruhovém pohybu.

Digitální výstupní signál se nastavuje/resetuje ve středu rohové dráhy pro flying body, jak je vidět na obrázku dole.

Obrázek ukazuje nastavení/reset digitálních výstupních signálů v rohové dráze s `MoveCDO`.



xx0500002215

U stop bodů doporučujeme použít „normální“ programovací sekvenci s `MoveC` + `SetDO`. Ale při používání stop bodu v instrukci `MoveCDO` se digitální výstupní signál nastavuje/resetuje, když robot dosáhne stop bodu.

Určený I/O signál se nastavuje/resetuje v režimu vykonávání nepřerušovaně a krokově dopředu, ale nikoliv krokově dozadu.

Omezení

Všeobecná omezení podle instrukce `MoveC`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_NO_ALIASIO_DEF jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.

ERR_NORUNUNIT jestliže není žádný kontakt s jednotkou I/O.

ERR_SIG_NOT_VALID jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
MoveCDO
  [ CirPoint ':=' ] < expression (IN) of robtarget > ','
  [ ToPoint ':=' ] < expression (IN) of robtarget > ','
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
  [ Signal ':=' ] < variable (VAR) of signaldo > ] ','
  [ Value ':=' ] < expression (IN) of dionum > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Posunout robot kruhově	*MoveC - Pohybuje robotem kruhově na str 358*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Pohyby s I/O nastaveními	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat TLoad, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.139 MoveCDO - Posouvá robot kruhově a nastavuje digitální výstup v rohu

RobotWare - OS

Pokračování

Pro informace o	Viz
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.140 MoveCGO - Posouvá robot kruhově a nastavuje skupinový výstupní signál v rohu

Použití

`MoveCGO` (*Move Circular Group Output*) se používá pro pohyb středového bodu nástroje (TCP) kruhově k dané destinaci. Určený skupinový výstupní signál se nastavuje do středu rohové dráhy u bodu určení. Během pohybu zůstává orientace normálně nezměněna ve vztahu ke kruhu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `MoveCGO`:

Příklad 1

```
MoveCGO p1, p2, v500, z30, tool2, go1 \Value:=5;
```

TCP nástroje `tool2`, je posunut kruhově na pozici `p2` s rychlostními daty `v500` a zónovými daty `z30`. Kruh je definován od pozice startu, bodu kruhu `p1` a cílového bodu `p2`. Skupinový výstupní signál `go1` je nastaven ve středu rohové dráhy u `p2`.

Argumenty

```
MoveCGO CirPoint ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal
    [\Value] | [\DValue] [\TLoad]
```

`CirPoint`

Datový typ: `robtarget`

Kruhový bod robotu. Kruhový bod je pozice na kruhu mezi start bodem a bodem určení. Aby bylo dosaženo největší přesnosti, měl by být umístěn asi na polovině vzdálenosti mezi body startu a určení. Jestliže je umístěn příliš blízko bodu startu nebo blízko bodu určení, robot může vydat varování. Bod kruhu je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci). Pozice externích os se nepoužívají.

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.140 MoveCGO - Posouvá robot kruhově a nastavuje skupinový výstupní signál v rohu
RobotWare - OS
Pokračování

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se pohybuje robot a externí osy. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného pozice.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadný systém objektu), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven kvůli kruhu ve vztahu k pracovnímu objektu, který bude proveden.

`Signal`

Datový typ: `signalgo`

Jméno skupinového výstupního signálu, který bude změněn.

`[ \Value ]`

Datový typ: `num`

Požadovaná hodnota signálu.

`[ \DValue ]`

Datový typ: `dnum`

Požadovaná hodnota signálu.

Jestliže není zapsán žádný z argumentů `\Value` nebo `\DValue`, zobrazí se chybová zpráva.

`[ \TLoad ]`

Total load

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

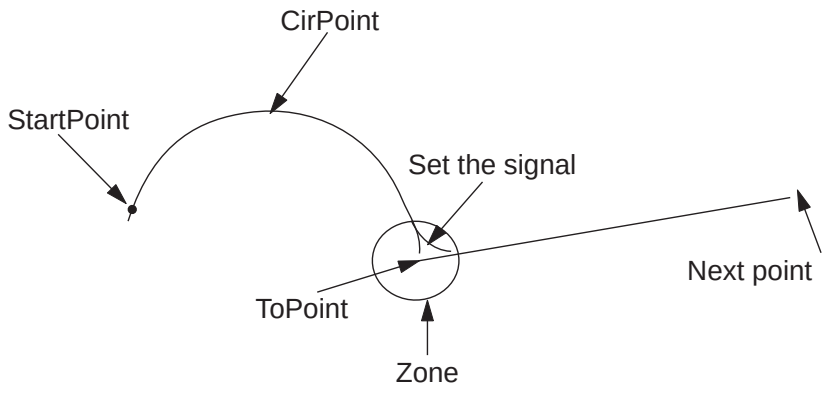
Vykonávání programu

V instrukci `MoveC` najdete více informací o kruhovém pohybu, *MoveC - Pohybuje robotem kruhově na str 358*.

Skupinový výstupní signál se nastavuje ve středu rohové dráhy pro flying body, jak je vidět na obrázku dole.

Obrázek ukazuje nastavení skupinového výstupního signálu v rohové dráze s `MoveCGO`.

```
MoveCGO p2, p2, v500, z30, tool2, go1 \Value:=5;
```



xx1400001116

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.140 MoveCGO - Posouvá robot kruhově a nastavuje skupinový výstupní signál v rohu

RobotWare - OS

Pokračování

U stop bodů doporučujeme použít „normální“ programovací sekvenci s `MoveC` a `SetGO`. Ale při používání stop bodu v instrukci `MoveCGO` se skupinový výstupní signál nastavuje, když robot dosáhne stop bodu.

Určený I/O signál se nastavuje v režimu vykonávání nepřerušovaně a krokově dopředu, ale nikoliv krokově dozadu.

Omezení

Všeobecná omezení podle instrukce `MoveC`, viz *MoveC - Pohybuje robotem kruhově na str 358*.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_GO_LIM`, jestliže argument `Value` nebo `DValue` pro určený analogový vstupní signál je mimo limity.

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
MoveCGO
  [ CirPoint ':=' ] < expression (IN) of robtarget > ','
  [ ToPoint ':=' ] < expression (IN) of robtarget > ','
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
  [ Signal ':=' ] < variable (VAR) of signalgo > ] ','
  [ '\' Value ':=' ] < expression (IN) of num > ]
  | [ '\' Dvalue' :=' ] < expression (IN) of dnum >
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Posunout robot kruhově	*MoveC - Pohybuje robotem kruhově na str 358*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Pohyby s I/O nastaveními	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O System, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.141 MoveCSync - Posunuje robot kruhově a vykonává proceduru RAPID

Použití

`MoveCSync` (*Move Circular Synchronously*) se používá pro pohyb středového bodu nástroje (TCP) kruhově k dané destinaci. Určené RAPID proceduře je přikázáno provedení ve středu rohové dráhy u bodu určení. Během pohybu zůstává orientace normálně nezměněna ve vztahu ke kruhu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `MoveCSync`:

Příklad 1

```
MoveCSync p1, p2, v500, z30, tool2, "proc1";
```

TCP nástroje `tool2`, je posunut kruhově na pozici `p2` s rychlostními daty `v500` a zónovými daty `z30`. Kruh je definován od pozice startu, bodu kruhu `p1` a bodu určení `p2`. Procedura `proc1` je vykonána ve středu rohové dráhy u `p2`.

Příklad 2

```
MoveCSync p1, p2, v500, z30, tool2, "MyModule:proc1";
```

Stejné, jako u příkladu 1 nahoře, ale zde bude lokálně deklarovaná procedura `proc1` v modulu `MyModule` volána ve středu rohové dráhy.

Argumenty

```
MoveCSync CirPoint ToPoint [\ID] Speed [\T] Zone Tool [\WObj]
    ProcName [\TLoad]
```

`CirPoint`

Datový typ: `robtarget`

Kruhový bod robotu. Kruhový bod je pozice na kruhu mezi start bodem a bodem určení. Aby bylo dosaženo největší přesnosti, měl by být umístěn asi na polovině vzdálenosti mezi body startu a určení. Jestliže je umístěn příliš blízko bodu startu nebo blízko bodu určení, robot může vydat varování. Bod kruhu je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci). Pozice externích os se nepoužívají.

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se pohybuje robot a externí osy. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného pozice.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadný systém objektu), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

`ProcName`

Procedure Name

Datový typ: `string`

Jméno RAPID procedury, která bude provedena u středu rohové dráhy v bodu destinace.

Procedura se bude provádět na úrovni TRAP (viz Vykonávání programu dole).

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.141 MoveCSync - Posunuje robot kruhově a vykonává proceduru RAPID

RobotWare - OS

Pokračování

Když je argument \TLoad nastaven na load0, potom není argument \TLoad uvažován a loaddata v aktuálním tooldata je použit místo toho.

Aby bylo možné použít argument \TLoad, je nezbytné nastavit hodnotu systémového parametru ModalPayLoadMode na 0. Jestliže ModalPayLoadMode je nastaven na 0, není dále možné používat instrukci GripLoad.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr ModalPayLoadMode nastaven na 0, operátor má možnost kopírovat loaddata z nástroje do existující nebo nové proměnné perzistentu loaddata při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užitečné zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému SimMode (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, loaddata ve volitelném argumentu \TLoad není uvažován a loaddata v aktuálním tooldata se použije místo toho.

POZNÁMKA

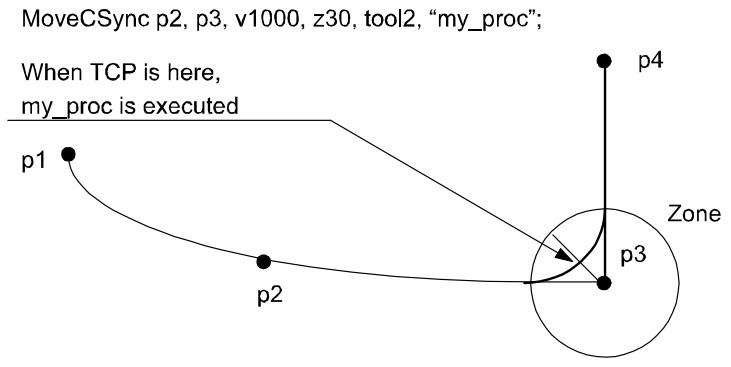
Výchozí funkčností pro řešení užitečné zátěže je použít instrukci GripLoad. Proto je výchozí hodnota systémového parametru ModalPayLoadMode 1.

Vykonávání programu

V instrukci MoveC najdete více informací o kruhových pohybech.

Určená RAPID procedura je přikázána k vykonání, když TCP dosahuje středu rohové dráhy v bodě destinace instrukce MoveCSync, jak je vidět na obrázku dole.

Obrázek ukazuje, že příkaz k vykonání uživatelsky definované RAPID procedury je proveden u středu rohové dráhy.



xx0500002216

Pro stop body doporučujeme použít „normální“ programovací sekvenci s MoveC + jiné RAPID instrukce v sekvenci.

Tabulka popisuje vykonávání určené RAPID procedury v různých režimech vykonávání:

Režim vykonávání	Vykonávání RAPID procedury
Nepřetržitě nebo cyklicky	Podle tohoto popisu

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Režim vykonávání	Vykonávání RAPID procedury
Dopředný krok	Ve stop bodu
Zpětný krok	Vůbec ne

`MoveCSync` je shrnutí (zapouzdření) instrukcí `TriggInt` a `TriggC`. Volání procedury je vykonáno na úrovni TRAP.

Jestliže středu rohové dráhy v destinačním bodě je dosaženo během zpomalování po zastavení programu, procedura nebude volána (vykonávání programu je zastaveno). Volání procedury bude vykonáno při novém spuštění programu.

Omezení

Všeobecná omezení podle instrukce `MoveC`.

Když robot dosáhne středu rohové dráhy, normálně následuje prodleva 2-30 ms, dokud není vykonána určená RAPID rutina v závislosti na typu pohybu, který je v té době prováděn.

Přepínání režimu vykonávání po zastavení programu z nepřetržitého na cyklický nebo krokový dopředu nebo dozadu má za následek chybu. Tyto chyba říká uživateli, že přepnutí režimu může mít za následek zmeškané vykonání RAPID procedury ve frontě na vykonávání na dráze.

Instrukci `MoveCSync` není možné používat na úrovni TRAP. Určenou RAPID proceduru není možné testovat s krokovým vykonáváním.

Syntaxe

```
MoveCSync
  [ CirPoint ':=' ] < expression (IN) of robtarget > ','
  [ ToPoint ':=' ] < expression (IN) of robtarget > ','
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
  [ ProcName ':=' ] < expression (IN) of string > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Pohybuje robotem kruhově	*MoveC - Pohybuje robotem kruhově na str 358*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.141 MoveCSync - Posunuje robot kruhově a vykonává proceduru RAPID

RobotWare - OS

Pokračování

Pro informace o	Viz
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Definuje přerušení se vztahem k pozici	*TriggInt - Definuje přerušení se vztahem k pozici na str 820*
Kruhový pohyb robotu s událostmi	*TriggC - Kruhový pohyb robotu s událostmi na str 796*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.142 MoveExtJ - Uvést do pohybu jednu nebo několik mechanických jednotek bez TCP

Použití

`MoveExtJ` (*Move External Joints*) se používá pro uvedení lineárních nebo rotačních externích os do pohybu. Externí osy mohou patřit jedné nebo několika mechanickým jednotkám bez TCP.

Tuto instrukci je možné používat pouze s aktuální programovou úlohou definovanou jako pohybová úloha (Motion Task) a jestliže úloha kontroluje jednu nebo několik mechanických jednotek bez TCP.

Základní příklady

Následující příklady názorně ukazují instrukci `MoveExtJ`:

Viz také *Další příklady na str 387*.

Příklad 1

```
MoveExtJ jpos10, vrot10, z50;
```

Posunout otočné externí osy do pozice spoje `jpos10` rychlostí `10` stupňů/sek. s daty zóny `z50`.

Příklad 2

```
MoveExtJ \Conc, jpos20, vrot10 \T:=5, fine \InPos:=inpos20;
```

Posunout externí osy do pozice spoje `jpos20` v 5. Vykonávání programu přejde okamžitě dopředu, ale externí osy se zastaví v pozici `jpos20` až do splnění konvergenčních kritérií v `inpos20`.

Argumenty

```
MoveExtJ [\Conc] ToJointPos [\ID] [\UseEOffs] Speed [\T] Zone
     [\Inpos]
```

`[ \Conc ]`

Concurrent

Datový typ: `switch`

Následné instrukce jsou vykonány během pohybu externí osy. Argument se obvykle nepoužívá, ale může se použít kvůli zabránění nechtěným zastavením způsobeným přetíženým CPU při používání fly-by bodů. To je vhodné, když naprogramované body jsou velmi blízko sebe při velkých rychlostech. Argument je také užitečný, když například komunikace s externím vybavením a synchronizace mezi externím vybavením a pohybem robotu nejsou žádoucí.

Použitím argumentu `\Conc` je počet pohybových instrukcí za sebou omezen na 5. V programové sekci, která obsahuje `StorePath-RestoPath`, nejsou pohybové instrukce s argumentem `\Conc` povoleny.

Jestliže tento argument je vypuštěn a `ToJointPos` není stop bodem, potom následná instrukce je vykonána předtím, než externí osy dosáhnou naprogramované zóny.

Tento argument se nemůže používat v koordinovaném synchronizovaném pohybu v systému MultiMove.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.142 MoveExtJ - Uvést do pohybu jednu nebo několik mechanických jednotek bez TCP

RobotWare - OS

Pokračování

`ToJointPos`

To Joint Position

Datový typ: `jointtarget`

Cílová absolutní pozice spoje externích os. Je definována jako jmenovitá pozice nebo uložena přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization ID

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`[ \UseEOffs ]`

Use External Offset

Datový typ: `switch`

Ofset pro externí osy, nastavený instrukcí `EOffsSet`, je aktivován pro instrukci `MoveExtJ`, když je použit argument `UseEOffs`. Více informací o externím ofsetu najdete v instrukci `EOffsSet`.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost lineárních nebo rotačních externích os.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se pohybují externí osy. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Data zóny pro pohyb. Data zóny definují stop bod nebo fly-by bod. Jestliže je to fly-by bod, potom velikost zóny popisuje zpomalení a zrychlení pro lineární nebo rotační externí osy.

`[ \Inpos ]`

In position

Datový typ: `stoppointdata`

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici externí osy ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Lineární nebo rotační externí osy jsou v pohybu k naprogramovanému bodu naprogramovanou rychlostí.

Další příklady

```
CONST jointtarget j1 :=
     [[9E9,9E9,9E9,9E9,9E9,9E9],[0,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget j2 :=
     [[9E9,9E9,9E9,9E9,9E9,9E9],[30,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget j3 :=
     [[9E9,9E9,9E9,9E9,9E9,9E9],[60,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget j4 :=
     [[9E9,9E9,9E9,9E9,9E9,9E9],[90,9E9,9E9,9E9,9E9,9E9]];
CONST speeddata rot_ax_speed := [0, 0, 0, 45];

MoveExtJ j1, rot_ax_speed, fine;
MoveExtJ j2, rot_ax_speed, z20;
MoveExtJ j3, rot_ax_speed, z20;
MoveExtJ j4, rot_ax_speed, fine;
```

V tomto příkladu je rotační jednoduchá osa posouvána k pozici spoje`0`, `30`, `60`, a `90` stupňů rychlostí `45` stupňů/sek.

Řešení chyb

Jestliže počet pohybových instrukcí za sebou pomocí argumentu `\Conc` byl překročen, potom je systémová proměnná `ERRNO` nastavena na `ERR_CONC_MAX`.

Tato chyba může být zpracována v chybovém handleru.

Syntaxe

```
MoveExtJ
  [ '\' Conc ',' ]
  [ ToJointPos ':=' ] < expression (IN) of jointtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ '\' UseEOffs ',' ]
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata >
  [ '\' Inpos ':=' < expression (IN) of stoppointdata >] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Definice jointtarget	*jointtarget - Data pozice svaru na str 1520*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souběžné vykonávání programu	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.143 MoveJ - Posunuje robot pohybem spoje

Použití

`MoveJ` se používá k rychlému přesunutí robotu z jednoho bodu do druhého, když tento pohyb nemusí být přímá linie.

Robot a externí osy se pohybují k cílové pozici podél nelineární dráhy. Všechny osy dosáhnout své cílové pozice současně.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `MoveJ`:

Viz také *Další příklady na str 391*.

Příklad 1

```
MoveJ p1, vmax, z30, tool2;
```

Středový bod nástroje (TCP) nástroje `tool2`se pohybuje podél nelineární dráhy k pozici `p1` s daty rychlosti `vmax` a daty zóny `z30`.

Příklad 2

```
MoveJ *, vmax \T:=5, fine, grip3;
```

TCP nástroje `grip3` se pohybuje podél nelineární dráhy ke stop bodu, který je uložen v instrukci (označeno s *). Celý pohyb trvá `5` sekund.

Argumenty

```
MoveJ [\Conc] ToPoint [\ID] Speed [\V] | [\T] Zone [\Z] [\Inpos]
     Tool [\WObj] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datový typ: `switch`

Následné instrukce jsou vykonány během pohybu robotu. Argument se obvykle nepoužívá, ale může se použít kvůli zabránění nechtěným zastavením způsobeným přetíženým CPU při používání fly-by bodů. To je vhodné, když naprogramované body jsou velmi blízko sebe při velkých rychlostech. Argument je také užitečný, když například komunikace s externím vybavením a synchronizace mezi externím vybavením a pohybem robotu nejsou žádoucí.

Použitím argumentu `\Conc` je počet pohybových instrukcí za sebou omezen na 5. V programové sekci, která obsahuje `StorePath-RestoPath`, nejsou pohybové instrukce s argumentem `\Conc` povoleny.

Jestliže tento argument je vypuštěn a `ToPoint` není stop bodem, následná instrukce je vykonána předtím, než robot dosáhne naprogramované zóny.

Tento argument se nemůže používat v koordinovaném synchronizovaném pohybu v systému MultiMove.

`ToPoint`

Datový typ: `robtarget`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID]`

Synchronization id

Datový typ: `identno`

Tento argument se musí použít v systému MultiMove, jestliže se jedná o koordinovaný synchronizovaný pohyb, a není povolen v žádném jiném případě.

Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Číslo id dává záruku, že pohyby se nepopletou za běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \V ]`

Velocity

Datový typ: `num`

Tento argument se používá k určení rychlosti TCP v mm/sek. přímo v instrukci. Je potom vyměněn za odpovídající rychlost, určenou v rychlostních datech.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`[ \Z ]`

Zone

Datový typ: `num`

Tento argument se používá k určení přesnosti pozice TCP robotu přímo v instrukci. Délka rohové dráhy je dána v mm, což je vyměněno za odpovídající zónu, která je určena v datech zóny.

`[ \Inpos ]`

In position

Datový typ: `stoppointdata`

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

`Tool`

Datový typ: `tooldata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.143 MoveJ - Posunuje robot pohybem spoje
RobotWare - OS
Pokračování

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného bodu - destinace.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

Středový bod nástroje je veden k bodu destinace s interpolací úhlů osy. To znamená, že každá osa je vedena konstantní rychlostí osy, a že všechny osy dosáhnou bodu destinace současně, což bude mít za následek nelineární dráhu.

Obecně řečeno, TCP je veden přibližnou naprogramovanou rychlostí (bez ohledu na to, jestli externí osy jsou koordinovány nebo nikoliv). Nástroj je reorientován a externí osy jsou v pohybu ve stejnou dobu, kdy se pohybuje TCP. Jestliže naprogramované rychlosti pro reorientaci nebo pro externí osy nemůže být dosaženo, rychlost TCP bude snížena.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Rohová dráha je obvykle generována, když je přenášen pohyb k další sekci dráhy. Jestliže stop bod je určen v datech zóny, vykonávání programu pokračuje pouze v případě, že robot a externí osy dosáhly příslušné pozice.

Další příklady

Více příkladů jak používat instrukci `MoveJ` je názorně uvedeno dole.

Příklad 1

```
MoveJ *, v2000\V:=2200, z40 \Z:=45, grip3;
```

TCP nástroje `grip3` je v pohybu podél nelineární dráhy k pozici uložené v instrukci. Pohyb je prováděn s daty nastavenými na `v2000` a `z40`. Rychlost a velikost zóny TCP jsou `2200` mm/sek. a `45` mm.

Příklad 2

```
MoveJ p5, v2000, fine \Inpos := inpos50, grip3;
```

TCP nástroje `grip3` je v pohybu v nelineární dráze ke stop bodu `p5`. Robot to považuje za bod, kde je splněno 50 % poziční podmínky a 50 % rychlostní podmínky pro stop bod `fine`. Čeká nejdéle 2 sekundy na splnění podmínek. Viz předdefinovaná data `inpos50` datového typu `stoppointdata`.

Příklad 3

```
MoveJ \Conc, *, v2000, z40, grip3;
```

TCP nástroje `grip3` je v pohybu podél nelineární dráhy k pozici uložené v instrukci. Následné logické instrukce jsou vykonány za pohybu robotu.

Příklad 4

```
MoveJ start, v2000, z40, grip3 \WObj:=fixture;
```

TCP nástroje `grip3` se pohybuje podél nelineární dráhy k pozici `start`. Tato pozice je určena v souřadném systému objektu pro `fixture`.

Řešení chyb

Jestliže počet pohybových instrukcí za sebou pomocí argumentu `\Conc` byl překročen, potom je systémová proměnná `ERRNO` nastavena na `ERR_CONC_MAX`.

Tato chyba může být zpracována v chybovém handleru.

Syntaxe

```
MoveJ
  [ '\' Conc ',' ]
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' V ':=' < expression (IN) of num > ]
  | [ '\' ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata >
  [ '\' Z ':=' < expression (IN) of num > ]
  [ '\' Inpos ':=' < expression (IN) of stoppointdata > ] ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.143 MoveJ - Posunuje robot pohybem spoje

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice dat stop bodu	*stoppointdata - Data stop bodu na str 1590*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Souběžné vykonávání programu	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.144 MoveJAO - Posouvá robot pohybem spoje a nastavuje analogový výstup v rohu

Použití

`MoveJAO` (*Move Joint Analog Output*) se používá k rychlému přesunutí robotu z jednoho bodu do druhého, když tento pohyb nemusí být přímá linie. Určený analogový výstupní signál je nastaven na střed rohové dráhy.

Robot a externí osy se pohybují k cílové pozici podél nelineární dráhy. Všechny osy dosáhnout své cílové pozice současně.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `MoveJAO`:

Příklad 1

```
MoveJAO p1, vmax, z30, tool2, ao1, 1.1;
```

Středový bod nástroje (TCP) nástroje `tool2` se pohybuje podél nelineární dráhy k pozici `p1` s daty rychlosti `vmax` a daty zóny `z30`. Výstup `ao1` je nastaven do středu rohové dráhy na `p1`.

Argumenty

```
MoveJAO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value
    [\TLoad]
```

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.144 MoveJAO - Posouvá robot pohybem spoje a nastavuje analogový výstup v rohu
RobotWare - OS
Pokračování

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného bodu - destinace.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

`Signal`

Datový typ: `signalao`

Jméno analogového výstupního signálu, který bude změněn.

`Value`

Datový typ: `num`

Požadovaná hodnota signálu.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

> **POZNÁMKA**
>
> Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

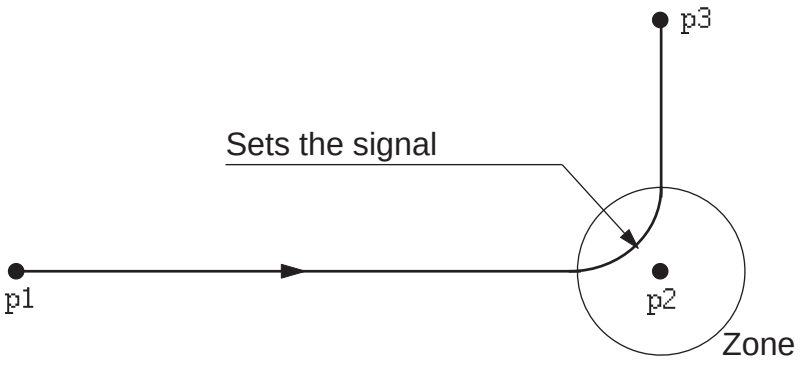
Vykonávání programu

V instrukci `MoveJ` najdete více informací o pohybu spoje, *MoveJ - Posunuje robot pohybem spoje na str 388*.

Analogový výstupní signál se nastavuje ve středu rohové dráhy pro flying body, jak je vidět na obrázku dole.

Obrázek ukazuje nastavení analogového výstupního signálu v rohové dráze s `MoveJAO`.

```
MoveJAO p2, vmax, z30, tool2, ao1, 1.1;
```



xx1400001118

U stop bodů doporučujeme použít „normální“ programovací sekvenci s `MoveJ` a `SetAO`. Ale při používání stop bodu v instrukci `MoveJAO` se analogový výstupní signál nastavuje, když robot dosáhne stop bodu.

Určený I/O signál se nastavuje v režimu vykonávání nepřerušovaně a krokově dopředu, ale nikoliv krokově dozadu.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_AO_LIM`, jestliže naprogramovaný argument `Value` pro určený analogový vstupní signál `Signal` je mimo limity.

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.144 MoveJAO - Posouvá robot pohybem spoje a nastavuje analogový výstup v rohu

RobotWare - OS

Pokračování

Syntaxe

```
MoveJAO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata>
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
  [ Signal ':=' ] < variable (VAR) of signalao>] ','
  [ Value ':=' ] < expression (IN) of num > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Posunuje robot pohybem spoje	*MoveJ - Posunuje robot pohybem spoje na str 388*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Pohyby s I/O nastaveními	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.145 MoveJDO - Posouvá robot pohybem spoje a nastavuje digitální výstup v rohu

Použití

`MoveJDO` (*Move Joint Digital Output*) se používá k rychlému přesunutí robotu z jednoho bodu do druhého, když tento pohyb nemusí být přímá linie. Určený analogový výstupní signál je nastaven/resetován na střed rohové dráhy.

Robot a externí osy se pohybují k cílové pozici podél nelineární dráhy. Všechny osy dosáhnout své cílové pozice současně.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `MoveJDO`:

Příklad 1

```
MoveJDO p1, vmax, z30, tool2, do1, 1;
```

Středový bod nástroje (TCP) nástroje `tool2` se pohybuje podél nelineární dráhy k pozici `p1` s daty rychlosti `vmax` a daty zóny `z30`. Výstup `do1` je nastaven do středu rohové dráhy na `p1`.

Argumenty

```
MoveJDO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value
    [\TLoad]
```

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.145 MoveJDO - Posouvá robot pohybem spoje a nastavuje digitální výstup v rohu
RobotWare - OS
Pokračování

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného bodu - destinace.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

`Signal`

Datový typ: `signaldo`

Jméno digitálního výstupního signálu, který bude změněn.

`Value`

Datový typ: `dionum`

Požadovaná hodnota signálu (0 nebo 1).

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

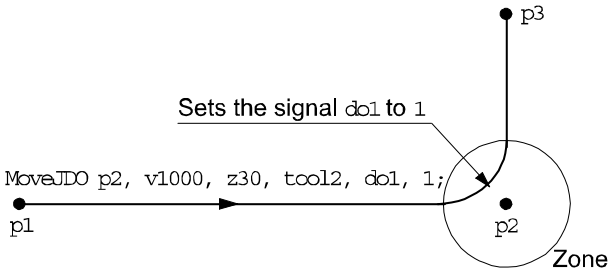
Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

V instrukci `MoveJ` najdete více informací pohybu spoje.

Digitální výstupní signál se nastavuje/resetuje ve středu rohové dráhy pro flying body, jak je vidět na obrázku dole.

Obrázek ukazuje nastavení/reset digitálních výstupních signálů v rohové dráze s `MoveJDO`.



xx0500002196

U stop bodů doporučujeme použít „normální“ programovací sekvenci s `MoveJ` + `SetDO`. Ale při používání stop bodu v instrukci `MoveJDO` se digitální výstupní signál nastavuje/resetuje, když robot dosáhne stop bodu.

Určený I/O signál se nastavuje/resetuje v režimu vykonávání nepřerušovaně a krokově dopředu, ale nikoliv krokově dozadu.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
MoveJDO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata>
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.145 MoveJDO - Posouvá robot pohybem spoje a nastavuje digitální výstup v rohu

RobotWare - OS

Pokračování

```
[ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
[ Signal ':=' ] < variable (VAR) of signaldo>] ','
[ Value ':=' ] < expression (IN) of dionum > ]
[ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Posunuje robot pohybem spoje	*MoveJ - Posunuje robot pohybem spoje na str 388*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Pohyby s I/O nastaveními	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.146 MoveJGO - Posouvá robot pohybem spoje a nastavuje skupinový výstupní signál v rohu

Použití

`MoveJGO` (*Move Joint Group Output*) se používá k rychlému přesunutí robotu z jednoho bodu do druhého, když tento pohyb nemusí být přímá linie. Určený skupinový výstupní signál je nastaven na střed rohové dráhy.

Robot a externí osy se pohybují k cílové pozici podél nelineární dráhy. Všechny osy dosáhnout své cílové pozice současně.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `MoveJGO`:

Příklad 1

```
MoveJGO p1, vmax, z30, tool2, go1 \Value:=5;
```

Středový bod nástroje (TCP) nástroje `tool2` se pohybuje podél nelineární dráhy k pozici `p1` s daty rychlosti `vmax` a daty zóny `z30`. Skupinový výstupní signál `go1` je nastaven do středu rohové dráhy na `p1`.

Argumenty

```
MoveJGO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal [\Value]
     | [\DValue] [\TLoad]
```

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.146 MoveJGO - Posouvá robot pohybem spoje a nastavuje skupinový výstupní signál v rohu
RobotWare - OS
Pokračování

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného bodu - destinace.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

`Signal`

Datový typ: `signalgo`

Jméno skupinového výstupního signálu, který bude změněn.

`[ \Value ]`

Datový typ: `num`

Požadovaná hodnota signálu.

`[ \DValue ]`

Datový typ: `dnum`

Požadovaná hodnota signálu.

Jestliže není zapsán žádný z argumentů `\Value` nebo `\DValue`, zobrazí se chybová zpráva.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

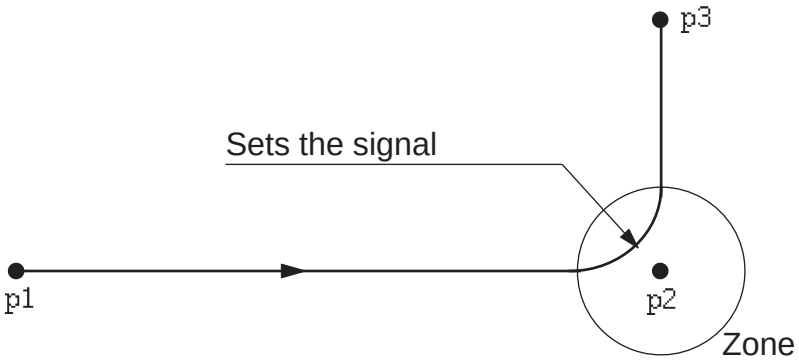
Vykonávání programu

V instrukci `MoveJ` najdete více informací pohybu spoje.

Skupinový výstupní signál se nastavuje ve středu rohové dráhy pro flying body, jak je vidět na obrázku dole.

Obrázek ukazuje nastavení skupinového výstupního signálu v rohové dráze s `MoveJGO`.

```
MoveJGO p2, vmax, z30, tool2, go1 \Value:=5;
```



xx1400001118

U stop bodů doporučujeme použít „normální“ programovací sekvenci s `MoveJ` a `SetGO`. Ale při používání stop bodu v instrukci `MoveJGO` se skupinový výstupní signál nastavuje, když robot dosáhne stop bodu.

Určený I/O signál se nastavuje v režimu vykonávání nepřerušovaně a krokově dopředu, ale nikoliv krokově dozadu.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_GO_LIM`, jestliže argument `Value` nebo `DValue` pro určený analogový vstupní signál je mimo limity.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.146 MoveJGO - Posouvá robot pohybem spoje a nastavuje skupinový výstupní signál v rohu

RobotWare - OS

Pokračování

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
MoveJGO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata>
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
  [ Signal ':=' ] < variable (VAR) of signalgo>] ','
  [ '\' Value ':=' ] < expression (IN) of num > ]
  | [ '\' Dvalue' :=' ] < expression (IN) of dnum >
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Posunuje robot pohybem spoje	*MoveJ - Posunuje robot pohybem spoje na str 388*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Pohyby s I/O nastaveními	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.147 MoveJSync - Posunuje robot pohybem spoje a vykonává proceduru RAPID.

Použití

`MoveJSync` (*Move Joint Synchronously*) se používá k rychlému přesunutí robotu z jednoho bodu do druhého, když tento pohyb nemusí být přímá linie. Určená RAPID procedura se přikazuje k vykonávání na středu rohové dráhy v bodu destinace.

Robot a externí osy se pohybují k cílové pozici podél nelineární dráhy. Všechny osy dosáhnout své cílové pozice současně.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `MoveJSync`:

Příklad 1

```
MoveJSync p1, vmax, z30, tool2, "proc1";
```

Středový bod nástroje (TCP) nástroje `tool2` se pohybuje podél nelineární dráhy k pozici `p1` s daty rychlosti `vmax` a daty zóny `z30`. Procedura `proc1` se vykonává ve středu rohové dráhy na `p1`.

Příklad 2

```
MoveJSync p1, vmax, z30, tool2, "MyModule:proc1";
```

Stejné, jako u příkladu 1 nahoře, ale zde bude lokálně deklarovaná procedura `proc1` v modulu `MyModule` volána ve středu rohové dráhy.

Argumenty

```
MoveJSync ToPoint [\ID] Speed [\T] Zone Tool [\WObj] ProcName
    [\TLoad]
```

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného bodu - destinace.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

`ProcName`

Procedure Name

Datový typ: `string`

Jméno procedury RAPID, která bude vykonána na středu rohové dráhy v bodu destinace. Volání procedury je volání s opožděnou vazbou a proto dědí vlastnosti.

Procedura se bude provádět na úrovni TRAP (viz Vykonávání programu dole).

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.147 MoveJSync - Posunuje robot pohybem spoje a vykonává proceduru RAPID.
RobotWare - OS
Pokračování

Je možné nechat program běžet zkušebně bez užitečné zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

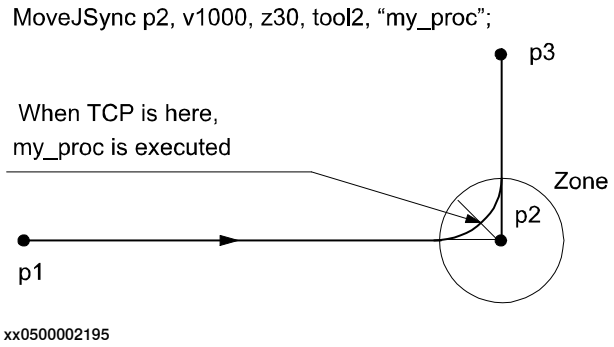
POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

V instrukci `MoveJ` najdete více informací o pohybech spojů.

Určená RAPID procedura je přikázána k vykonání, když TCP dosahuje středu rohové dráhy v bodě destinace instrukce `MoveJSync`, jak je vidět na obrázku dole.



xx0500002195

Pro stop body doporučujeme použít „normální“ programovací sekvenci s `MoveJ` + jiné RAPID instrukce v sekvenci.

Tabulka popisuje vykonávání určené RAPID procedury v různých režimech vykonávání:

Režim vykonávání	Vykonávání RAPID procedury
Nepřetržitě nebo cyklicky	Podle tohoto popisu
Dopředný krok	Ve stop bodu
Zpětný krok	Vůbec ne

`MoveJSync` je shrnutí (zapouzdření) instrukcí `TriggInt` a `TriggJ`. Volání procedury je vykonáno na úrovni TRAP.

Jestliže středu rohové dráhy v destinačním bodě je dosaženo během zpomalování po zastavení programu, procedura nebude volána (vykonávání programu je zastaveno). Volání procedury bude vykonáno při novém spuštění programu.

Omezení

Když robot dosáhne středu rohové dráhy, normálně následuje prodleva 2-30 ms, dokud není vykonána určená `RAPID` rutina v závislosti na typu pohybu, který je v té době prováděn.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.147 MoveJSync - Posunuje robot pohybem spoje a vykonává proceduru RAPID.

Přepínání režimu vykonávání po zastavení programu z nepřetržitého na cyklický nebo krokový dopředu nebo dozadu má za následek chybu. Tyto chyba říká uživateli, že přepnutí režimu může mít za následek zmeškané vykonání RAPID procedury ve frontě na vykonávání na dráze.

Instrukci `MoveJSync` není možné používat na úrovni TRAP. Určenou RAPID proceduru není možné testovat s krokovým vykonáváním.

Syntaxe

```
MoveJSync
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >] ','
  [ Speed ':=' ] < expression (IN) of speeddata >
  '\' T ':=' < expression (IN) of num >] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj '=' < persistent (PERS) of wobjdata > ] ','
  [ ProcName '=' ] < expression (IN) of string > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Posunuje robot pohybem spoje	*MoveJ - Posunuje robot pohybem spoje na str 388*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Definuje přerušení se vztahem k pozici	*TriggInt - Definuje přerušení se vztahem k pozici na str 820*
Pohyby robotu podle osy s událostmi	*TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.147 MoveJSync - Posunuje robot pohybem spoje a vykonává proceduru RAPID.

RobotWare - OS

Pokračování

Pro informace o	Viz
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.148 MoveL - Pohybuje robotem lineárně

Použití

`MoveL` se používá k posunutí středního bodu nástroje (TCP) lineárně k dané destinaci. Jestliže TCP má zůstat stacionární, potom se tato instrukce může použít také pro reorientaci nástroje.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `MoveL`:

Viz také *Další příklady na str 414*.

Příklad 1

```
MoveL p1, v1000, z30, tool2;
```

TCP nástroje `tool2` je posunuto lineárně k pozici `p1` s daty rychlosti `v1000` a daty zóny `z30`.

Příklad 2

```
MoveL *, v1000\T:=5, fine, grip3;
```

TCP nástroje `grip3` se pohybuje lineárně ke stop bodu, který je uložen v instrukci (označeno s *). Celý pohyb trvá `5` sekund.

Argumenty

```
MoveL [\Conc] ToPoint [\ID] Speed [\V] | [ \T] Zone [\Z] [\Inpos]
     Tool [\WObj] [\Corr] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datový typ: `switch`

Následné instrukce jsou vykonány během pohybu robotu. Argument se obvykle nepoužívá, ale může se použít kvůli zabránění nechtěným zastavením způsobeným přetíženým CPU při používání fly-by bodů. To je vhodné, když naprogramované body jsou velmi blízko sebe při velkých rychlostech. Argument je také užitečný, když například komunikace s externím vybavením a synchronizace mezi externím vybavením a pohybem robotu nejsou žádoucí.

Použitím argumentu `\Conc` je počet pohybových instrukcí za sebou omezen na 5. V programové sekci, která obsahuje `StorePath-RestoPath`, nejsou pohybové instrukce s argumentem `\Conc` povoleny.

Jestliže tento argument je vypuštěn a `ToPoint` není stop bodem, následná instrukce je vykonána předtím, než robot dosáhne naprogramované zóny.

Tento argument se nemůže používat v koordinovaném synchronizovaném pohybu v systému MultiMove.

`ToPoint`

Datový typ: `robtarget`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.148 MoveL - Pohybuje robotem lineárně
RobotWare - OS
Pokračování

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \V ]`

Velocity

Datový typ: `num`

Tento argument se používá k určení rychlosti TCP v mm/sek. přímo v instrukci. Je potom vyměněn za odpovídající rychlost, určenou v rychlostních datech.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`[ \Z ]`

Zone

Datový typ: `num`

Tento argument se používá k určení přesnosti pozice TCP robotu přímo v instrukci. Délka rohové dráhy je dána v mm, což je vyměněno za odpovídající zónu, která je určena v datech zóny.

`[ \Inpos ]`

In position

Datový typ: `stoppointdata`

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

`Tool`

Datový typ: `tooldata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného bodu - destinace.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární nástroj nebo koordinované externí osy, tento argument musí být potom určen k provedení lineárního pohybu ve vztahu k pracovnímu objektu.

`[ \Corr ]`

Correction

Datový typ: `switch`

Korekční data zapsaná do vstupu korekcí instrukcí `CorrWrite` budou přidána k pozici dráhy a destinace, jestliže tento argument je přítomen.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.148 MoveL - Pohybuje robotem lineárně
RobotWare - OS
Pokračování

Vykonávání programu

Robot a externí jednotky jsou posunovány k pozici destinace následovně:

- TCP nástroje je posunováno lineárně konstantní naprogramovanou rychlostí.
- Nástroj je reorientován ve stejných intervalech podél dráhy.
- Nekoordinované externí osy jsou vykonávány konstantní rychlostí tak, aby dorazily do bodu určení (destinace) současně s osami robotu.

Jestliže není možné dosáhnout naprogramované rychlosti pro reorientaci nebo pro externí osy, rychlost TCP bude snížena.

Rohová dráha je obvykle generována, když je přenášen pohyb k další sekci dráhy. Jestliže stop bod je určen v datech zóny, vykonávání programu pokračuje pouze v případě, že robot a externí osy dosáhly příslušné pozice.

Další příklady

Více příkladů jak používat instrukci `MoveL` je názorně uvedeno dole.

Příklad 1

```
MoveL *, v2000 \V:=2200, z40 \Z:=45, grip3;
```

TCP nástroje `grip3` je v pohybu lineárně k pozici uložené v instrukci. Pohyb je prováděn s daty nastavenými na `v2000` a `z40`. Rychlost a velikost zóny TCP jsou `2200` mm/sek. a `45` mm.

Příklad 2

```
MoveL p5, v2000, fine \Inpos := inpos50, grip3;
```

TCP nástroje `grip3` je v pohybu kruhově ke stop bodu `p5`. Robot to považuje za bod, kde je splněno 50 % poziční podmínky a 50 % rychlostní podmínky pro stop bod `fine`. Čeká nejdéle 2 sekundy na splnění podmínek. Viz předdefinovaná data `inpos50` datového typu `stoppointdata`..

Příklad 3

```
MoveL \Conc, *, v2000, z40, grip3;
```

TCP nástroje `grip3` je v pohybu lineárně k pozici uložené v instrukci. Následné logické instrukce jsou vykonány za pohybu robotu.

Příklad 4

```
MoveL start, v2000, z40, grip3 \WObj:=fixture;
```

TCP nástroje `grip3` se pohybuje podél lineárně k pozici `start`. Tato pozice je určena v souřadném systému objektu pro `fixture`.

Příklad s TLoad

```
MoveL p1, v1000, fine, tool2;
! Pick up the payload
Set gripperdo;
MoveL p2, v1000, z30, tool2 \TLoad:=tool2piece;
MoveL p3, v1000, fine, tool2 \TLoad:=tool2piece;
! Release the payload
Reset gripperdo;
MoveL p4, v1000, fine, tool2;
```

TCP nástroje `tool2` se pohybuje lineárně k pozici p1, kde je nabrána užitečná zátěž. Od této pozice je TCP posunut na pozici p2 a p3 pomocí celkové zátěže

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`tool2piece`. `loaddata` v aktuálním `tooldata` není bráno v úvahu. Užitečná zátěž je uvolněna a při pohybu k p4 je zátěž nástroje vzata opět v úvahu.

Řešení chyb

Jestliže počet pohybových instrukcí za sebou pomocí argumentu `\Conc` byl překročen, potom je systémová proměnná `ERRNO` nastavena na `ERR_CONC_MAX`.

Tato chyba může být zpracována v chybovém handleru.

Syntaxe

```
MoveL
  [ '\' Conc ',' ]
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >] ','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' V ':=' < expression (IN) of num > ]
  | [ '\' T ':=' < expression (IN) of num > ] ','
  [Zone ':=' ] < expression (IN) of zonedata >
  [ '\' Z ':='< expression (IN) of num > ]
  [ '\' Inpos ':=' < expression (IN) of stoppointdata > ] ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ]
  [ '\' Corr ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice dat stop bodu	*stoppointdata - Data stop bodu na str 1590*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Zapisuje do vstupu korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Souběžné vykonávání programu	*Technická referenční příručka - Přehled RAPID*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.148 MoveL - Pohybuje robotem lineárně

RobotWare - OS

Pokračování

Pro informace o	Viz
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.149 MoveLAO - Posouvá robot lineárně a nastavuje analogový výstup v rohu

Použití

`MoveLAO` (*Move Linearly Analog Output*) se používá k přesunutí středového bodu nástroje (TCP) lineárně k dané destinaci. Určený analogový výstupní signál je nastaven na střed rohové dráhy.

Když TCP má zůstat stacionární, potom je možné tuto instrukci použít také k reorientaci nástroje.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `MoveLAO`:

Příklad 1

```
MoveLAO p1, v1000, z30, tool2, ao1, 1.1;
```

Středový bod nástroje (TCP) nástroje `tool2` se pohybuje lineárně k pozici `p1` s daty rychlosti `v1000` a daty zóny `z30`. Výstup `ao1` je nastaven do středu rohové dráhy na `p1`.

Argumenty

```
MoveLAO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value
    [\TLoad]
```

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.149 MoveLAO - Posouvá robot lineárně a nastavuje analogový výstup v rohu
RobotWare - OS
Pokračování

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného bodu - destinace.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

`Signal`

Datový typ: `signalao`

Jméno analogového výstupního signálu, který bude změněn.

`Value`

Datový typ: `num`

Požadovaná hodnota signálu.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

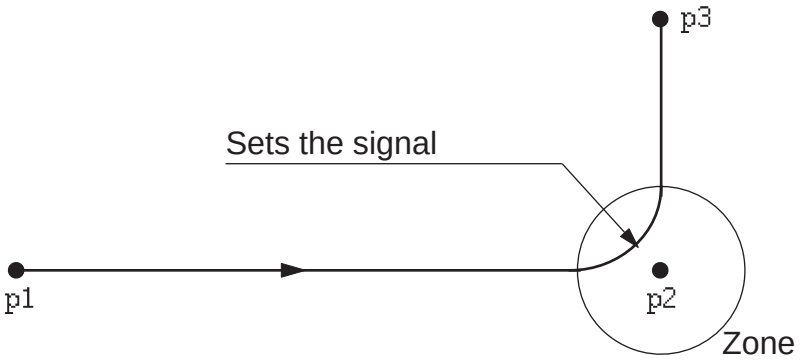
Vykonávání programu

V instrukci `MoveL` najdete více informací o lineárních pohybech.

Analogový výstupní signál se nastavuje ve středu rohové dráhy pro flying body, jak je vidět na obrázku dole.

Obrázek ukazuje nastavení analogového výstupního signálu v rohové dráze s `MoveLAO`.

```
MoveLAO p2, v1000, z30, tool2, ao1, 1.1;
```



xx1400001118

U stop bodů doporučujeme použít „normální“ programovací sekvenci s `MoveL` a `SetAO`. Ale při používání stop bodu v instrukci `MoveLAO` se analogový výstupní signál nastavuje, když robot dosáhne stop bodu.

Určený I/O signál se nastavuje v režimu vykonávání nepřerušovaně a krokově dopředu, ale nikoliv krokově dozadu.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_AO_LIM`, jestliže naprogramovaný argument `Value` pro určený analogový vstupní signál `Signal` je mimo limity.

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.149 MoveLAO - Posouvá robot lineárně a nastavuje analogový výstup v rohu

RobotWare - OS

Pokračování

Syntaxe

```
MoveLAO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' ] < persistent (PERS) of wobjdata > ','
  [ Signal ':=' ] < variable (VAR) of signalao >] ','
  [ Value ':=' ] < expression (IN) of num > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Pohybuje robotem lineárně	*MoveL - Pohybuje robotem lineárně na str 411*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Pohyby s I/O nastaveními	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.150 MoveLDO - Posouvá robot lineárně a nastavuje digitální výstup v rohu

Použití

`MoveLDO` (*Move Linearly Digital Output*) se používá k přesunutí středového bodu nástroje (TCP) lineárně k dané destinaci. Určený digitální výstupní signál je nastaven/resetován na střed rohové dráhy.

Když TCP má zůstat stacionární, potom je možné tuto instrukci použít také k reorientaci nástroje.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `MoveLDO`:

Příklad 1

```
MoveLDO p1, v1000, z30, tool2, do1,1;
```

Středový bod nástroje (TCP) nástroje `tool2` se pohybuje lineárně k pozici `p1` s daty rychlosti `v1000` a daty zóny `z30`. Výstup do1 je nastaven do středu rohové dráhy na `p1`.

Argumenty

```
MoveLDO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value
    [\TLoad]
```

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.150 MoveLDO - Posouvá robot lineárně a nastavuje digitální výstup v rohu
RobotWare - OS
Pokračování

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného bodu - destinace.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

`Signal`

Datový typ: `signaldo`

Jméno digitálního výstupního signálu, který bude změněn.

`Value`

Datový typ: `dionum`

Požadovaná hodnota signálu (0 nebo 1).

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

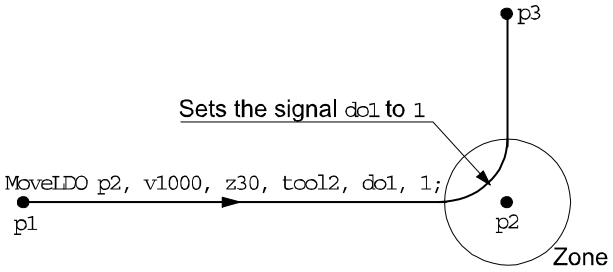
Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

V instrukci `MoveL` najdete více informací o lineárních pohybech.

Digitální výstupní signál se nastavuje/resetuje ve středu rohové dráhy pro flying body, jak je vidět na obrázku dole.

Obrázek ukazuje nastavení/reset digitálních výstupních signálů v rohové dráze s `MoveLDO`.



xx0500002193

U stop bodů doporučujeme použít „normální“ programovací sekvenci s `MoveL` + `SetDO`. Ale při používání stop bodu v instrukci `MoveLDO` se digitální výstupní signál nastavuje/resetuje, když robot dosáhne stop bodu.

Určený I/O signál se nastavuje/resetuje v režimu vykonávání nepřerušovaně a krokově dopředu, ale nikoliv krokově dozadu.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
MoveLDO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.150 MoveLDO - Posouvá robot lineárně a nastavuje digitální výstup v rohu

RobotWare - OS

Pokračování

```
[ '\' WObj ':=' ] < persistent (PERS) of wobjdata > ','
[ Signal ':=' ] < variable (VAR) of signaldo >] ','
[ Value ':=' ] < expression (IN) of dionum > ]
[ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Pohybuje robotem lineárně	*MoveL - Pohybuje robotem lineárně na str 411*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Pohyby s I/O nastaveními	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.151 MoveLGO - Posouvá robot lineárně a nastavuje skupinový výstupní signál v rohu

Použití

`MoveLGO` (*Move Linearly Group Output*) se používá k přesunutí středového bodu nástroje (TCP) lineárně k dané destinaci. Určený skupinový výstupní signál je nastaven na střed rohové dráhy.

Když TCP má zůstat stacionární, potom je možné tuto instrukci použít také k reorientaci nástroje.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `MoveLGO`:

Příklad 1

```
MoveLGO p1, v1000, z30, tool2, go1 \Value:=5;
```

Středový bod nástroje (TCP) nástroje `tool2` se pohybuje lineárně k pozici `p1` s daty rychlosti `v1000` a daty zóny `z30`. Skupinový výstupní signál `go1` je nastaven do středu rohové dráhy na `p1`.

Argumenty

```
MoveLGO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal [\Value]
     | [\DValue] [\TLoad]
```

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného bodu - destinace.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

`Signal`

Datový typ: `signalgo`

Jméno skupinového výstupního signálu, který bude změněn.

`[ \Value ]`

Datový typ: `num`

Požadovaná hodnota signálu.

`[ \DValue ]`

Datový typ: `dnum`

Požadovaná hodnota signálu.

Jestliže není zapsán žádný z argumentů `\Value` nebo `\DValue`, zobrazí se chybová zpráva.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

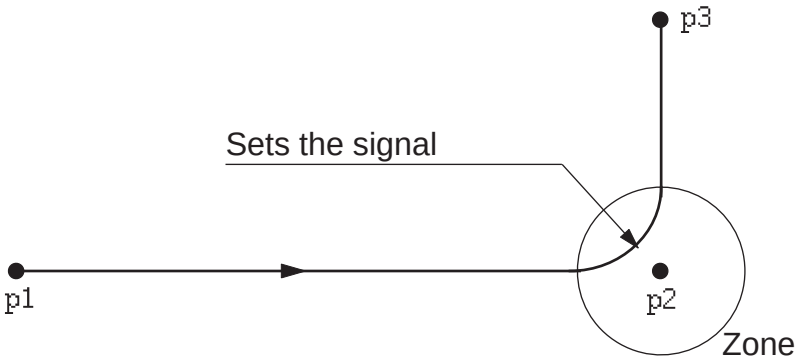
Vykonávání programu

V instrukci `MoveL` najdete více informací o lineárních pohybech.

Analogový výstupní signál se nastavuje ve středu rohové dráhy pro flying body, jak je vidět na obrázku dole.

Obrázek ukazuje nastavení skupinového výstupního signálu v rohové dráze s `MoveLGO`.

```
MoveLGO p2, v1000, z30, tool2, go1 \Value:=5;
```



xx1400001118

U stop bodů doporučujeme použít „normální“ programovací sekvenci s `MoveL` a `SetGO`. Ale při používání stop bodu v instrukci `MoveLGO` se skupinový výstupní signál nastavuje, když robot dosáhne stop bodu.

Určený I/O signál se nastavuje v režimu vykonávání nepřerušovaně a krokově dopředu, ale nikoliv krokově dozadu.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.151 MoveLGO - Posouvá robot lineárně a nastavuje skupinový výstupní signál v rohu

RobotWare - OS

Pokračování

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
MoveLGO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' ] < persistent (PERS) of wobjdata > ','
  [ Signal ':=' ] < variable (VAR) of signaldo >] ','
  [ '\' Value ':=' ] < expression (IN) of num > ]
  | [ '\' Dvalue' :=' ] < expression (IN) of dnum >
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Pohybuje robotem lineárně	*MoveL - Pohybuje robotem lineárně na str 411*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Pohyby s I/O nastaveními	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.152 MoveLSync - Posunuje robot lineárně a vykonává proceduru RAPID

Použití

`MoveLSync` (*Move Linearly Synchronously*) se používá k přesunutí středového bodu nástroje (TCP) lineárně k dané destinaci. Určená procedura RAPID se přikazuje a vykonává u středu rohové dráhy v bodu destinace.

Když TCP má zůstat stacionární, potom je možné tuto instrukci použít také k reorientaci nástroje.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `MoveLSync`:

Příklad 1

```
MoveLSync p1, v1000, z30, tool2, "proc1";
```

Středový bod nástroje (TCP) nástroje `tool2` se pohybuje lineárně k pozici `p1` s daty rychlosti `v1000` a daty zóny `z30`. Procedura `proc1` se vykonává ve středu rohové dráhy na `p1`.

Příklad 2

```
MoveLSync p1, v1000, z30, tool2, "proc1";
```

Stejné, jako u příkladu 1 nahoře, ale zde bude lokálně deklarovaná procedura `proc1` v modulu `MyModule` volána ve středu rohové dráhy.

Argumenty

```
MoveLSync ToPoint [\ID] Speed [\T] Zone Tool [\WObj] ProcName
    [\TLoad]
```

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.152 MoveLSync - Posunuje robot lineárně a vykonává proceduru RAPID

RobotWare - OS

Pokračování

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určeného bodu - destinace.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

`ProcName`

Procedure Name

Datový typ: `string`

Jméno procedury RAPID, která bude vykonána na středu rohové dráhy v bodu destinace. Volání procedury je volání s opožděnou vazbou a proto dědí vlastnosti.

Procedura se bude provádět na úrovni TRAP (viz Vykonávání programu dole).

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Je možné nechat program běžet zkušebně bez užitečné zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

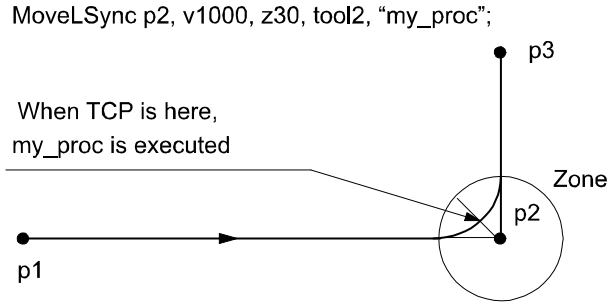
Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

V instrukci `MoveL` najdete více informací o lineárních pohybech.

Určená RAPID procedura je přikázána k vykonání, když TCP dosahuje středu rohové dráhy v bodě destinace instrukce `MoveLSync`, jak je vidět na obrázku dole.

Obrázek ukazuje, že příkaz k vykonání uživatelsky definované RAPID procedury je proveden ve středu rohové dráhy.



xx0500002194

Pro stop body doporučujeme použít „normální“ programovací sekvenci s `MoveL` + jiné RAPID instrukce v sekvenci.

Tabulka popisuje vykonávání určené RAPID procedury v různých režimech vykonávání:

Režim vykonávání:	Vykonávání RAPID procedury
Nepřetržitě nebo cyklicky	Podle tohoto popisu
Dopředný krok	Ve stop bodu
Zpětný krok	Vůbec ne

`MoveLSync` je shrnutí (zapouzdření) instrukcí `TriggInt` a `TriggL`. Volání procedury je vykonáno na úrovni TRAP.

Jestliže středu rohové dráhy v destinačním bodě je dosaženo během zpomalování po zastavení programu, procedura nebude volána (vykonávání programu je zastaveno). Volání procedury bude vykonáno při novém spuštění programu.

Omezení

Když robot dosáhne středu rohové dráhy, normálně následuje prodleva 2-30 ms, dokud není vykonána určená RAPID rutina v závislosti na typu pohybu, který je v té době prováděn.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.152 MoveLSync - Posunuje robot lineárně a vykonává proceduru RAPID

RobotWare - OS

Pokračování

Přepínání režimu vykonávání po zastavení programu z nepřetržitého na cyklický nebo krokový dopředu nebo dozadu má za následek chybu. Tyto chyba říká uživateli, že přepnutí režimu může mít za následek zmeškané vykonání RAPID procedury ve frontě na vykonávání na dráze.

Instrukci `MoveLSync` není možné používat na úrovni TRAP. Určenou RAPID proceduru není možné testovat s krokovým vykonáváním.

Syntaxe

```
MoveLSync
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
  [ ProcName ':=' ] < expression (IN) of string > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Pohybuje robotem lineárně	*MoveL - Pohybuje robotem lineárně na str 411*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Definuje přerušení se vztahem k pozici	*TriggInt - Definuje přerušení se vztahem k pozici na str 820*
Lineární pohyby robotu s událostmi	*TriggL - Lineární pohyby robotu s událostmi na str 839*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.153 MToolRotCalib - Kalibrace rotace pro pohybující se nástroj

Použití

`MToolRotCalib` *(Moving Tool Rotation Calibration)* se používá pro kalibrování otáčení pohybujícího se nástroje.

Pozice robotu a jeho pohyby mají vždy vztah k jeho souřadnému systému nástroje, tj. TCP a orientaci nástroje. Abychom získali největší přesnost, je důležité definovat souřadný systém nástroje tak přesně, jak je to možné.

Kalibraci je možné provést také ruční metodou pomocí FlexPendantu (popsáno v *Provozní příručka - IRC5 s FlexPendant*em, sekce *Programování a testování*).

Popis

Abyste mohli definovat orientaci nástroje, potřebujete světově pevně stanovený hrot v pracovním prostoru robotu.

Před použitím instrukce `MToolRotCalib` musí být splněny některé předpoklady:

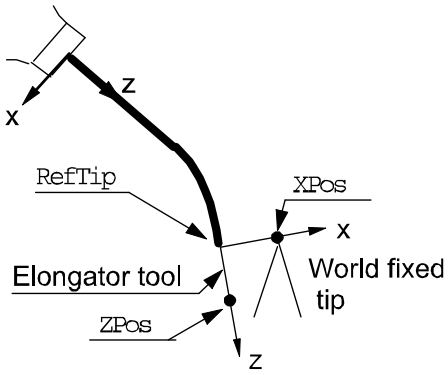
- Nástroj, který se má kalibrovat, musí být upevněn na robotu a definován se správným komponentem `robhold(TRUE)`.
- Při používání robotu s absolutní přesností by měla být již definována zátěž a těžiště pro nástroj. `LoadIdentify` se může použít pro definici zátěže.
- TCP hodnota nástroje musí být již definována. Kalibraci je možné provést s instrukcí `MToolTCPCalib`..
- `tool0`, `wobj0`, a `PDispOff` musí být aktivovány před ručním posunem (jogging) robotu.
- Ručně posuňte (jog) TCP aktuálního nástroje co nejblíže ke světově pevně stanovenému hrotu (počátek souřadného systému nástroje) a definujte `jointtarget` pro referenční bod `RefTip`.
- Ručně posuňte robot (jog) bez změny nástroje, aby světově pevně stanovený hrot ukazoval na některý bod na kladné z-ose souřadného systému nástroje, a definujte `jointtarget` pro `ZPos`. bodu.
- Případně ručně posuňte robot (jog) bez změny nástroje, aby světově pevně stanovený hrot ukazoval na některý bod na kladné x-ose souřadného systému nástroje, a definujte `jointtarget` pro `XPos`. bodu.

Jako pomoc pro směrování kladné z-osy a x-osy můžete použít některý typ prodlužovacího nástroje.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Na obrázku dole vidíte definici jointtarget pro `RefTip`, `ZPos` a volitelnou `XPos`.



xx0500002192

POZNÁMKA

Nedoporučuje se modifikovat pozice `RefTip`, `ZPos`, a `XPos` v instrukci `MToolRotCalib`.

Základní příklady

Následující příklady názorně ukazují instrukci `MToolRotCalib`:

Příklad 1

```
! Created with the world fixed tip pointing at origin, positive
! z-axis, and positive x-axis of the wanted tool coordinate
! system.
CONST jointtarget pos_tip := [...];
CONST jointtarget pos_z := [...];
CONST jointtarget pos_x := [...];

PERS tooldata tool1:= [ TRUE, [[20, 30, 100], [1, 0, 0 ,0]], [0.001,
     [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0]];

! Instructions for creating or ModPos of pos_tip, pos_z, and pos_x
MoveAbsJ pos_tip, v10, fine, tool0;
MoveAbsJ pos_z, v10, fine, tool0;
MoveAbsJ pos_x, v10, fine, tool0;

! Only tool calibration in the z direction
MToolRotCalib pos_tip, pos_z, tool1;
```

Orientace nástroje (`tframe.rot`) ve směru Z `tool1` je vypočítána. Směry X a Y orientace nástroje jsou vypočítány, aby se shodovaly se souřadným systémem zápěstí.

Příklad 2

```
! Calibration with complete tool orientation
MToolRotCalib pos_tip, pos_z \XPos:=pos_x, tool1;
```

Kompletní orientace nástroje (`tframe.rot`) od `tool1` je vypočítána.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.153 MToolRotCalib - Kalibrace rotace pro pohybující se nástroj

RobotWare - OS

Pokračování

Argumenty

```
MToolRotCalib RefTip ZPos [\XPos]Tool
```

`RefTip`

Datový typ: `jointtarget`

Bod, kde TCP nástroje ukazuje na světově pevný hrot.

`ZPos`

Datový typ: `jointtarget`

Bod prodlužovače, který definuje kladný směr z.

`[\XPos]`

Datový typ: `jointtarget`

Bod prodlužovače, který definuje kladný směr x. Jestliže tento bod je vypuštěn, potom se směry x a y nástroje budou krýt s odpovídajícími osami v souřadném systému zápěstí.

`Tool`

Datový typ: `tooldata`

Proměnná perzistentu nástroje, který má být kalibrován.

Vykonávání programu

Systém vypočítává a aktualizuje orientaci nástroje (`tframe.rot`) v určeném tooldata. Výpočet je založen na určených 2 nebo 3 `jointtarget`. Zbývající data v tooldata jako je TCP (`tframe.trans`) se nemění.

Syntaxe

```
MToolRotCalib
  [ RefTip ':=' ] < expression (IN) of jointtarget > ','
  [ ZPos ':=' ] < expression (IN) of jointtarget >
  [ '\' XPos ':=' < expression (IN) of jointtarget > ] ','
  [ Tool ':=' ] < persistent (PERS) of tooldata > ';'
```

Související informace

Pro informace o	Viz
Kalibrace TCP pro pohyblivý nástroj	*MToolTCPCalib - Kalibrace TCP pro pohybující se nástroj na str 437*
Kalibrace TCP pro stacionární nástroj	*SToolTCPCalib - Kalibrace TCP pro stacionární nástroj na str 728*
Kalibrace TCP a rotace pro stacionární nástroj	*SToolRotCalib - Kalibrace TCP a rotace pro stacionární nástroj na str 725*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.154 MToolTCPCalib - Kalibrace TCP pro pohybující se nástroj

Použití

`MToolTCPCalib` (*Moving Tool TCP Calibration*) se používá pro kalibrování TCP pro pohyblivý nástroj.

Pozice robotu a jeho pohyby mají vždy vztah k jeho souřadnému systému nástroje, tj. TCP a orientaci nástroje. Abychom získali největší přesnost, je důležité definovat souřadný systém nástroje tak přesně, jak je to možné.

Kalibraci je možné provést také ruční metodou pomocí FlexPendantu (popsáno v *Provozní příručka - IRC5 s FlexPendantem*, sekce *Programování a testování*).

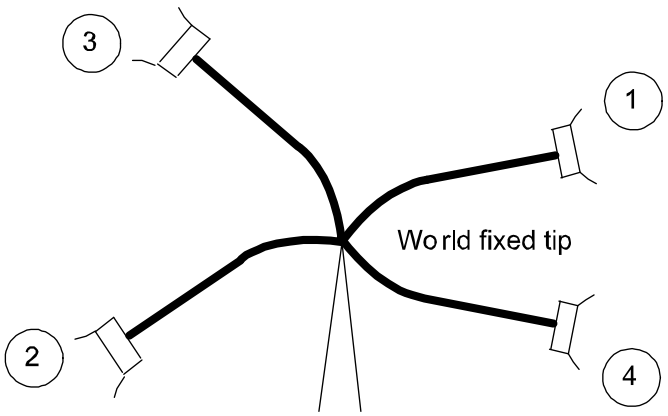
Popis

Abyste mohli definovat TCP nástroje, potřebujete světově pevně stanovený hrot v pracovním prostoru robotu.

Před použitím instrukce `MToolTCPCalib` musí být splněny některé předpoklady:

- Nástroj, který se má kalibrovat, musí být upevněn na robotu a definován se správným komponentem `robholdTRUE`.
- Při používání robotu s absolutní přesností by měla být již definována zátěž a těžiště pro nástroj. `LoadIdentify` se může použít pro definici zátěže.
- `tool0`, `wobj0`, a`PDispOff` musí být aktivovány před ručním posunem (jogging) robotu.
- Ručně posuňte (jog) TCP aktuálního nástroje co nejblíže ke světově pevně stanovenému hrotu a definujte `jointtarget` pro první bod `p1`.
- Definujte další tři pozice (`p2`, `p3`, a `p4`), všechny s různými orientacemi.

Definice 4 joittarget p1....p4, viz obrázek dole.



xx0500002191

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.154 MToolTCPCalib - Kalibrace TCP pro pohybující se nástroj

RobotWare - OS

Pokračování

> **POZNÁMKA**
>
> Nedoporučuje se modifikovat pozice `Pos1` až `Pos4` v instrukci `MToolTCPCalib`.
>
> Reorientace mezi 4 pozicemi by měla být co největší, stavějící robot do odlišných konfigurací. Je také dobrou praxí kontrolovat kvalitu TCP po kalibraci. Což se může provést reorientací nástroje kvůli kontrole, jestli TCP je v klidu.

Základní příklady

Následující příklad názorně ukazuje instrukci `MToolTCPCalib`:

Příklad 1

```
! Created with actual TCP pointing at the world fixed tip
CONST jointtarget p1 := [...];
CONST jointtarget p2 := [...];
CONST jointtarget p3 := [...];
CONST jointtarget p4 := [...];

PERS tooldata tool1:= [TRUE, [[0, 0, 0], [1, 0, 0 ,0]], [0.001,
     [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0]];
VAR num max_err;
VAR num mean_err;
...
! Instructions for createing or ModPos of p1 - p4
MoveAbsJ p1, v10, fine, tool0;
MoveAbsJ p2, v10, fine, tool0;
MoveAbsJ p3, v10, fine, tool0;
MoveAbsJ p4, v10, fine, tool0;
...
MToolTCPCalib p1, p2, p3, p4, tool1, max_err, mean_err;
```

Hodnota TCP (`tframe.trans`) od `tool1` bude kalibrována a aktualizována. `max_err` a `mean_err` budou uchovávat max. chybu v mm od vypočítaného TCP a střední chybu v mm od vypočítaného TCP.

Argumenty

```
MToolTCPCalib Pos1 Pos2 Pos3 Pos4 Tool MaxErr MeanErr
```

`Pos1`

Datový typ: `jointtarget`

Bod prvního přiblížení.

`Pos2`

Datový typ: `jointtarget`

Bod druhého přiblížení.

`Pos3`

Datový typ: `jointtarget`

Bod třetího přiblížení.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Pos4`

Datový typ: `jointtarget`

Bod čtvrtého přiblížení.

`Tool`

Datový typ: `tooldata`

Proměnná perzistentu nástroje, který má být kalibrován.

`MaxErr`

Datový typ: `num`

Max chyba v mm na jeden bod přiblížení.

`MeanErr`

Datový typ: `num`

Průměrná vzdálenost bodů přiblížení od vypočítaného TCP, tj. jak přesně byl robot polohován ve vztahu k hrotu.

Vykonávání programu

Systém vypočítává a aktualizuje hodnotu TCP v souřadném systému zápěstí (`tframe.trans`) v určeném `tooldata`. Výpočet je založen na určených 4 `jointtarget`. Zbývající data v tooldata, jako je orientace nástroje`tframe.rot`), se nemění.

Syntaxe

```
MToolTCPCalib
  [ Pos1 ':=' ] < expression (IN) of jointtarget > ','
  [ Pos2 ':=' ] < expression (IN) of jointtarget > ','
  [ Pos3 ':=' ] < expression (IN) of jointtarget > ','
  [ Pos4 ':=' ] < expression (IN) of jointtarget > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata > ','
  [ MaxErr ':=' ] < variable (VAR) of num > ','
  [ MeanErr ':=' ] < variable (VAR) of num > ';'
```

Související informace

Pro informace o	Viz
Kalibrace rotace pro pohyblivý nástroj	*MToolRotCalib - Kalibrace rotace pro pohybující se nástroj na str 434*
Kalibrace TCP pro stacionární nástroj	*SToolTCPCalib - Kalibrace TCP pro stacionární nástroj na str 728*
Kalibrace TCP a rotace pro stacionární nástroj	*SToolRotCalib - Kalibrace TCP a rotace pro stacionární nástroj na str 725*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.155 Open - Otevírá soubor nebo sériový kanál

Použití

`Open` se používá k otevření souboru nebo sériového kanálu pro čtení a zápis.

Základní příklady

Následující příklady názorně ukazují instrukci `Open`:

Viz také *Další příklady na str 442*.

Příklad 1

```
VAR iodev logfile;
...
Open "HOME:" \File:= "LOGFILE1.DOC", logfile \Write;
```

Soubor `LOGFILE1.DOC` v jednotce `HOME:` je otevřen pro zápis. Jméno reference `logfile` se použije později v programu při zápisu do souboru.

Příklad 2

```
VAR iodev logfile;
...
Open "LOGFILE1.DOC", logfile \Write;
```

Stejný výsledek jako u příkladu 1. Výchozím adresářem je `HOME:`.

Argumenty

```
Open Object [\File] IODevice [\Read] | [\Write] | [\Append] [\Bin]
```

`Object`

Datový typ: `string`

I/O objekt (I/O zařízení), který má být otevřen, např. `"HOME:"`, `"TEMP:"`, `"com1:"` nebo `"pc:"`(doplněk).

Tabulka popisuje různá I/O zařízení na řadiči robotu.

Jméno I/O zařízení	Typ I/O zařízení
`"HOME:"` nebo `diskhome`[i]	SD karta
`"TEMP:"` or `disktemp`[i]	SD karta
`"RemovableDisk1:"` or `usbdisk1`[i] `"RemovableDisk2:"` or `usbdisk2`[i] `"RemovableDisk3:"` or `usbdisk3`[i]	např. USB paměťové médium
`"com1:"`[ii]	Sériový kanál
`"pc:"`[iii]	Namontovaný disk
`"RAMDISK:"` or `diskram`[i, iv]	Disková paměť RAM

[i] Řetězec RAPID definující jméno zařízení.
[ii] Jméno uživatelsky definovaného sériového kanálu definované v systémových parametrech.
[iii] Aplikační protokol, cesta serveru definovaná v parametrech systému.
[iv] Disková paměť RAM neslouží pro stálé uložení dat. Kapacita je kolem 100 MB a je vymazána při každém vypnutí.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Následující tabulka popisuje různá I/O zařízení na virtuálním řadiči.

Jméno I/O zařízení	Typ I/O zařízení
"HOME:" nebo diskhome[i]	
"TEMP:" nebo disktemp	Pevný disk
"RemovableDisk1:" nebo usbdisk1 "RemovableDisk2:" nebo usbdisk2 "RemovableDisk3:" nebo usbdisk3	např. USB paměťové médium
"RAMDISK:" nebo diskram[i]	Pevný disk ..\TEMP (ukazuje k TEMP složce, která je umístěna ve stejné složce jako váš virtuální systém)

[i] Řetězec RAPID definující jméno zařízení.

[\File]

Datový typ: string

Jméno souboru, který má být otevřen, např. "LOGFILE1.DOC" nebo "LOGDIR/LOGFILE1.DOC"

Kompletní cesta může být určena také v argumentu Object, "HOME:/LOGDIR/LOGFILE.DOC".

IODevice

Datový typ: iodev

Reference k souboru nebo sériovému kanálu, který má být otevřen. Tato reference se potom použije pro čtení nebo zápis do souboru nebo sériového kanálu.

[\Read]

Datový typ: switch

Otevírá soubor nebo sériový kanál pro čtení. Při čtení ze souboru čtení začíná od začátku souboru.

[\Write]

Datový typ: switch

Otevírá soubor nebo sériový kanál pro zápis. Jestliže zvolený soubor už existuje, potom je jeho obsah vymazán. Cokoliv se následně zapíše, je zapsáno na začátek souboru.

[\Append]

Datový typ: switch

Otevírá soubor nebo sériový kanál pro zápis. Jestliže zvolený soubor už existuje, potom cokoliv se následně zapíše, je zapsáno na konec souboru.

Otevřete soubor nebo sériový kanál s \Append a bez argumentů \Bin. Instrukce otevírá soubor založený na znacích nebo sériový kanál pro zápis.

Otevřete soubor nebo sériový kanál s argumenty \Append a \Bin. Instrukce otevírá binární soubor nebo sériový kanál pro čtení i zápis. Argumenty \Read, \Write, \Append se vzájemně vylučují. Jestli žádný z nich není určen, potom instrukce funguje stejným způsobem jako argument \Write pro soubory založené na znacích

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.155 Open - Otevírá soubor nebo sériový kanál
RobotWare - OS
Pokračování

nebo sériový kanál (instrukce bez argumentu \Bin) a stejným způsobem jako argument \Append pro binární soubory nebo sériový kanál (instrukce s argumentem \Bin).

[\Bin]

Datový typ: switch

Soubor nebo sériový kanál se otevírá v binárním režimu. Jestliže žádný z argumentů \Read, \Write nebo \Append není určen, potom instrukce otevírá binární soubor nebo sériový kanál pro čtení i zápis, s ukazatelem souboru na konci souboru.

Instrukce Rewind se může v případě potřeby používat k nastavení ukazatele souboru na začátek souboru.

Sada instrukcí k přístupu k binárnímu souboru nebo sériovému kanálu je odlišná od sady instrukcí k přístupu k souboru založenému na znacích.

Další příklady

Více příkladů jak používat instrukci Open je názorně uvedeno dole.

Příklad 1

```
VAR iodev printer;
...
Open "com1:", printer \Bin;
WriteStrBin printer, "This is a message to the printer\0D";
Close printer;
```

Sériový kanál com1: je otevřen pro binární čtení a zápis. Jméno reference printer se použije později při zápisu a uzavírání sériového kanálu.

Příklad 2

```
VAR iodev io_device;
VAR rawbytes raw_data_out;
VAR rawbytes raw_data_in;
VAR num float := 0.2;
VAR string answer;

ClearRawBytes raw_data_out;
PackDNHeader "10", "20 1D 24 01 30 64", raw_data_out;
PackRawBytes float, raw_data_out, (RawBytesLen(raw_data_out)+1)
     \Float4;

Open "/FCI1:/dsqc328_1", io_device \Bin;
WriteRawBytes io_device, raw_data_out;
ReadRawBytes io_device, raw_data_in \Time:=1;
Close io_device;

UnpackRawBytes raw_data_in, 1, answer \ASCII:=10;
```

V tomto příkladu je raw_data_out vyčištěno a potom zabaleno s hlavičkou DeviceNet a float s hodnotou 0.2.

Zařízení "/FCI1/:dsqc328_1" je otevřeno a aktuálně platná data v raw_data_out jsou zapsána do zařízení. Potom program čeká nejdéle 1 sek. na čtení ze zařízení, což je uloženo do raw_data_in.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Po zavření zařízení "/FCI1/:dsqc328_1" jsou přečtená data rozbalena jako řetězec 10 znaků a uložena do řetězce s názvem answer.

Vykonávání programu

Určený soubor nebo sériový kanál jsou otevřeny, takže je možné z nich číst nebo do nich zapisovat.

Je možné několikrát otevřít stejný fyzický soubor ve stejnou dobu, ale každá výzva instrukce Open vrátí odlišnou referenci k souboru (datový typ iodev). Například je možné mít jeden ukazatel zápisu a jeden odlišný ukazatel čtení ke stejnému souboru ve stejném čase.

Proměnná iodev použitá při otevírání souboru nebo sériového kanálu musí být volná. Jestliže byla otevřena před tím k otevření souboru, tento soubor musí být zavřen před vydáním nové instrukce Open se stejnou proměnnou iodev.

Při zastavení programu a posunutí PP na Main bude zavřen každý otevřený soubor nebo sériový kanál v programové úloze a I/O popisovač v proměnné typu iodev bude resetován. Výjimkou pravidla jsou proměnné, které jsou instalovány sdíleně v systému typu globální VAR nebo LOCAL VAR. Takový soubor nebo sériový kanál patřící k celému systému bude stále otevřen.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu iodev bude resetován.

Řešení chyb

Jestliže není možné otevřít soubor, potom je systémová proměnná ERRNO nastavena na ERR_FILEOPEN. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
Open
  [ Object ':='] <expression (IN) of string>
  [ '\' File ':=' <expression (IN) of string>] ','
  [ IODevice ':='] <variable (VAR) of iodev>
  [ '\' Read] |
  [ '\' Write] |
  [ '\' Append]
  [ '\' Bin] ';'
```

Související informace

Pro informace o	Viz
Zápis, čtení a zavírání souborů nebo sériových kanálů	*Technická referenční příručka - Přehled RAPID*
Rozhraní příkazů aplikační sběrnice Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.156 OpenDir - Otevřít adresář

Použití

`OpenDir` se používá k otevření adresáře pro další průzkum.

Základní příklady

Následující příklad názorně ukazuje instrukci `OpenDir`:

Příklad 1

```
PROC lsdir(string dirname)
  VAR dir directory;
  VAR string filename;
  OpenDir directory, dirname;
  WHILE ReadDir(directory, filename) DO
    TPWrite filename;
  ENDWHILE
  CloseDir directory;
ENDPROC
```

Tento příklad vytiskne jména všech souborů nebo podadresářů pod určeným adresářem.

Argumenty

```
OpenDir Dev Path
```

`Dev`

Datový typ: `dir`

Proměnná s referencí k adresáři získanému od `OpenDir`. Tato proměnná se potom použije pro čtení z adresáře.

`Path`

Datový typ: `string`

Cesta k adresáři.

Omezení

Otevřené adresáře by měl uživatel vždy po čtení zavřít (instrukce `CloseDir`).

Řešení chyb

Jestliže cesta ukazuje k neexistujícímu adresáři nebo když je otevřeno mnoho adresářů současně, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
OpenDir
  [ Dev ':=' ] < variable (VAR) of dir> ','
  [ Path ':=' ] < expression (IN) of string> ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Adresář	*dir - Struktura adresáře souborů na str 1484*
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Načíst adresář	*ReadDir - Přečíst další vstupní data v adresáři na str 1283*
Zavřít adresář	*CloseDir - Zavřít adresář na str 125*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.157 PackDNHeader - Zabalit hlavičku DeviceNet do dat rawbytes

Použití

`PackDNHeader` se používá pro zabalení hlavičky explicitní zprávy DeviceNet do kontejneru typu `rawbytes`.

Datová část zprávy DeviceNet může být potom nastavena s instrukcí `PackRawBytes`.

Základní příklady

Následující příklady názorně ukazují instrukci `PackDNHeader`:

Příklad 1

```
VAR rawbytes raw_data;

PackDNHeader "0E", "6,20 01 24 01 30 06,9,4", raw_data;
```

Zabalte hlavičku pro explicitní zprávu DeviceNet se servisním kódem `"0E"` a řetězcem cesty `"6,2001 24 01 30 06,9,4"` do `raw_data` odpovídajících získání sériového čísla od některé I/O jednotky.

Tato zpráva je připravena k odeslání bez doplňování zprávy dalšími daty.

Příklad 2

```
VAR rawbytes raw_data;

PackDNHeader "10", "20 1D 24 01 30 64", raw_data;
```

Zabalte hlavičku pro explicitní zprávu DeviceNet se servisním kódem `"10"` a řetězcem cesty `"201D 24 01 30 64"` do `raw_data` odpovídajících nastavení času filtru pro stoupající hranu na insignal 1 u některé I/O jednotky.

Tato zpráva musí být rozšířena o data pro čas filtru. To je možné provést s instrukcí `PackRawBytes` se začátkem u indexu `RawBytesLen(raw_data)+1` (provedeno po `PackDNHeader`).

Argumenty

```
PackDNHeader Service Path RawData
```

`Service`

Datový typ: `string`

Služba, která bude provedena, jako je získání nebo nastavení atributu. Bude určeno v šestnáctkovém kódu v řetězci např. `"IF"`.

Délka řetězce	2 znaky
Formát	'0' -' 9', 'a' -'f', 'A' - 'F'
Rozsah	"00" - "FF"

Hodnoty pro `Service` jsou v souboru EDS. Více popisů najdete v Open DeviceNet Vendor Association *ODVA DeviceNet Specification revision 2.0*.

`Path`

Datový typ: `string`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Hodnoty pro `Path` jsou v souboru EDS. Více popisů najdete v Open DeviceNet Vendor Association *ODVA DeviceNet Specification revision 2.0*.

Podpora pro dlouhý formát řetězce (např. "`6,20 1D 24 01 30 64,8,1`") a krátký formát řetězce (např. "`20 1D 24 01 30 64`").

`RawData`

Datový typ: `rawbytes`

Kontejner proměnné, který bude zabalen s daty hlavičky zprávy u indexu 1 v `RawData`.

Vykonávání programu

Během vykonávání programu je kontejner `RawData` zprávy DeviceNet:

- nejprve úplně vyčištěn
- a potom je část hlavičky zabalena s daty

Formátovat hlavičku DeviceNet

Instrukce `PackDNHeader` vytvoří hlavičku zprávy DeviceNet s následujícím formátem:

RawData Header-Format	Žádné bajty	Poznámka
Formát	1	Interní kód IRC5 pro DeviceNet
Servis	1	Šestnáctkový kód pro službu
Velikost cesty	1	v bajtech
Cesta	x	Znaky ASCII

Datová část zprávy DeviceNet může být potom nastavena s instrukcí `PackRawBytes` začínající u indexu získaného s (`RawBytesLen(my_rawdata)+1`).

Syntaxe

```
PackDNHeader
  [ Service ':=' ] < expression (IN) of string> ','
  [ Path ':=' ] < expression (IN) of string> ','
  [ RawData ':=' ] < variable (VAR) of rawbytes> ';'
```

Související informace

Pro informace o	Viz
Data`rawbytes`	*rawbytes - Data raw na str 1559*
Získat délku dat `rawbytes`	*RawBytesLen - Získat délku dat rawbytes na str 1278*
Vyčistit obsah dat `rawbytes`	*ClearRawBytes - Vyčistit obsahy rawbyte dat na str 118*
Kopírovat obsah dat `rawbytes`	*CopyRawBytes - Kopírovat obsah dat rawbytes na str 137*
Zabalit data do dat `rawbytes`	*PackRawBytes - Zabalit data do dat rawbytes na str 449*
Zapsat data `rawbytes`	*WriteRawBytes - Zapsat data rawbytes na str 994*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.157 PackDNHeader - Zabalit hlavičku DeviceNet do dat rawbytes

RobotWare - OS

Pokračování

Pro informace o	Viz
Načíst data `rawbytes`	*ReadRawBytes - Přečíst data rawbytes na str 530*
Rozbalit data z dat `rawbytes`	*UnpackRawBytes - Rozbalit data z rawbyte dat na str 908*
Funkce Bit/Bajt	*Technická referenční příručka - Přehled RAPID*
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.158 PackRawBytes - Zabalit data do dat rawbytes

Použití

`PackRawBytes` se používá pro zabalení obsahu proměnných typu `num`, `dnum`, `byte`, nebo `string` do kontejneru typu `rawbytes`.

Základní příklady

Následující příklad názorně ukazuje instrukci `PackRawBytes`:

```
VAR rawbytes raw_data;
VAR num integer := 8;
VAR dnum bigInt := 4294967295;
VAR num float := 13.4;
VAR byte data1 := 122;
VAR byte byte1;
VAR string string1:="abcdefg";
PackDNHeader "10", "20 1D 24 01 30 64", raw_data;
```

Zabalit hlavičku pro DeviceNet do `raw_data`.

Potom zabalte požadovaná data aplikační sběrnice do `raw_data` s `PackRawBytes`. Příklad dole ukazuje, jak se mohou přidávat různá data.

Příklad 1

```
PackRawBytes integer, raw_data, (RawBytesLen(raw_data)+1) \IntX :=
     DINT;
```

Obsah dalších 4 bajtů po hlavičce v `raw_data` bude `8` decimální.

Příklad 2

```
PackRawBytes bigInt, raw_data, (RawBytesLen(raw_data)+1) \IntX :=
     UDINT;
```

Obsah dalších 4 bajtů po hlavičce v `raw_data` bude `4294967295` decimální.

Příklad 3

```
PackRawBytes bigInt, raw_data, (RawBytesLen(raw_data)+1) \IntX :=
     LINT;
```

Obsah dalších 8 bajtů po hlavičce v `raw_data` bude `4294967295` decimální.

Příklad 4

```
PackRawBytes float, raw_data, RawBytesLen(raw_data)+1) \Float4;
```

Obsah dalších 4 bajtů v `raw_data` bude 13.4 decimální.

Příklad 5

```
PackRawBytes data1, raw_data, (RawBytesLen(raw_data)+1) \ASCII;
```

Obsah dalšího bajtu v `raw_data` bude `122`, ASCII kód pro "z".

Příklad 6

```
PackRawBytes string1, raw_data, (RawBytesLen(raw_data)+1) \ASCII;
```

Obsah dalších 7 bajtů v `raw_data` bude `"abcdefg"`, kódováno v ASCII.

Příklad 7

```
byte1 := StrToByte("1F" \Hex);
PackRawBytes byte1, raw_data, (RawBytesLen(raw_data)+1) \Hex1;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.158 PackRawBytes - Zabalit data do dat rawbytes

RobotWare - OS

Pokračování

Obsah dalšího bajtu v `raw_data` bude `"1F"`, šestnáctkový.

Argumenty

```
PackRawBytes Value RawData [ \Network ] StartIndex [\Hex1]|[\IntX
     ] | [ \Float4 ] | [ \ASCII ]
```

`Value`

Datový typ: `anytype`

Data, která budou zabalena do `RawData`.

Přípustné datové typy: `num`, `dnum`, `byte`, nebo `string`. Pole není možné používat.

`RawData`

Datový typ: `rawbytes`

Kontejner proměnné, který bude zabalen s daty.

`[ \Network ]`

Datový typ: `switch`

Indikuje, že `integer` a `float` bude zabaleno do big endian (pořadí sítě) reprezentace v `RawData`. ProfiBus a InterBus používají big endian.

Bez tohoto přepínače bude `integer` a `float` zabalen v little endian (nikoliv pořadí sítě) reprezentaci v `RawData`. DeviceNet používá little endian.

Pouze relevantní společně s volitelným parametrem `\IntX - UINT`, `UDINT`, `INT`, `DINT` a `\Float4`.

`StartIndex`

Datový typ: `num`

`StartIndex` mezi 1 a 1024 indikuje, kam bude první bajt, obsažený v `Value`, umístěn v `RawData`.

`[ \Hex1 ]`

Datový typ: `switch`

`Value`, která bude zabalena, má formát `byte` a bude převedena na šestnáctkový formát a uložena v 1 bajtu v `RawData`.

`[ \IntX ]`

Datový typ: `inttypes`

`Value`, která bude zabalena, má formát `num` nebo `dnum`. Je to celé číslo a bude uloženo v `RawData` podle této určené konstanty datového typu `inttypes`.

Viz část *Předdefinovaná data na str 451*.

`[ \Float4 ]`

Datový typ: `switch`

`Value`, která bude zabalena, má formát `num` a bude uložena jako plovoucí, 4 bajty, v `RawData`.

`[ \ASCII ]`

Datový typ: `switch`

`Value`, která bude zabalena, má formát `byte` nebo `string`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže `Value`, která bude zabalena, má formát `byte`, bude uložena v `RawData` jako 1 bajt a bude interpretovat `Value` jako ASCII kód pro znak.

Jestliže `Value`, která bude uložena, má formát řetězce (1-80 znaků), potom bude uložena do `RawData` jako znaky ASCII se stejným počtem znaků, jako je obsaženo v `Value`. Data řetězce nejsou ukončena `NULL` systémem v datech typu `rawbytes`. Je na programátorovi, aby podle potřeby doplnil hlavičku řetězce (vyžaduje se pro DeviceNet).

Musí být naprogramován jeden z argumentů `\Hex1`, `\IntX`, `\Float4`, nebo `\ASCII` .

Jsou dovoleny následující kombinace:

Datový typ Value (Hodnota):	Dovolené volitelné parametry:
`num` *)	`\IntX`
`dnum` **)	`\IntX`
`num`	`\Float4`
`string`	`\ASCII` (1-80 znaků)
`byte`	`\Hex1 \ASCII`ob

*) Musí být celé číslo v rozsahu hodnoty zvolené symbolické konstanty `USINT`, `UINT`, `UDINT`, `SINT`, `INT` nebo `DINT`.

**) Musí být celé číslo v rozsahu hodnoty zvolené symbolické konstanty `USINT`, `UINT`, `UDINT`, `ULINT`, `SINT`, `INT`, `DINT` nebo `LINT`.

Vykonávání programu

Během vykonávání programu jsou data zabalena od proměnné typu `anytype` do kontejneru typu `rawbytes`.

Aktuální délka platných bajtů v proměnné `RawData` je nastavena na:

- (`StartIndex` + packed_number_of_bytes - 1) (zabalený počet bajtů - 1)
- Aktuální délka platných bajtů v proměnné `RawData` se nezmění, jestliže kompletní balicí operace je provedena uvnitř staré aktuální délky platných bajtů v proměnné `RawData`.

Předdefinovaná data

Následující symbolické konstanty datového typu `inttypes` jsou předdefinovány a mohou se používat k určení celého čísla v argumentu `\IntX`.

Symbolická konstanta	Konstantní hodnota	Formát celého čísla	Rozsah hodnoty celého čísla
`USINT`	1	Nepodepsané 1-bajtové celé číslo	0 ... 255
`UINT`	2	Nepodepsané 2-bajtové celé číslo	0 ... 65 535
`UDINT`	4	Nepodepsané 4-bajtové celé číslo	0 ... 8 388 608 *) 0 ... 4 294 967 295 ****)
`ULINT`	8	Nepodepsané 8-bajtové celé číslo	0 ... 4 503 599 627 370 496**)

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.158 PackRawBytes - Zabalit data do dat rawbytes

RobotWare - OS

Pokračování

Symbolická konstanta	Konstantní hodnota	Formát celého čísla	Rozsah hodnoty celého čísla
SINT	- 1	Podepsané 1-bajtové celé číslo	- 128... 127
INT	- 2	Podepsané 2-bajtové celé číslo	- 32 768 ... 32 767
DINT	- 4	Podepsané 4-bajtové celé číslo	- 8 388 607 ... 8 388 608 *) -2 147 483 648 ... 2 147 483 647 ***)
LINT	- 8	Podepsané 8-bajtové celé číslo	- 4 503 599 627 370 496... 4 503 599 627 370 496 **)

*) RAPID omezení pro ukládání celého čísla v datovém typu num.

**) RAPID omezení pro ukládání celého čísla v datovém typu dnum.

***) Rozsah při používání proměnné dnum a inttype DINT.

****) Rozsah při používání proměnné dnum a inttype UDINT.

Syntaxe

```
PackRawBytes
  [ Value ':=' ] < expression (IN) of anytype> ','
  [ RawData ':=' ] < variable (VAR) of rawbytes>
  [ '\' Network ] ','
  [ StartIndex ':=' ] < expression (IN) of num>
  [ '\' Hex1 ]
  | [ '\' IntX ':=' < expression (IN) of inttypes>]
  | [ '\' Float4 ]
  | [ '\' ASCII] ';'
```

Související informace

Pro informace o	Viz
Datarawbytes	*rawbytes - Data raw na str 1559*
Získat délku dat rawbytes	*RawBytesLen - Získat délku dat rawbytes na str 1278*
Vyčistit obsah dat rawbytes	*ClearRawBytes - Vyčistit obsahy rawbyte dat na str 118*
Kopírovat obsah dat rawbytes	*CopyRawBytes - Kopírovat obsah dat rawbytes na str 137*
Zabalit hlavičku DeviceNet do dat rawbytes	*PackDNHeader - Zabalit hlavičku DeviceNet do dat rawbytes na str 446*
Zapsat data rawbytes	*WriteRawBytes - Zapsat data rawbytes na str 994*
Načíst data rawbytes	*ReadRawBytes - Přečíst data rawbytes na str 530*
Rozbalit data z dat rawbytes	*UnpackRawBytes - Rozbalit data z rawbyte dat na str 908*
Funkce Bit/Bajt	*Technická referenční příručka - Přehled RAPID*
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.159 PathAccLim - Omezit TCP zrychlení podél dráhy

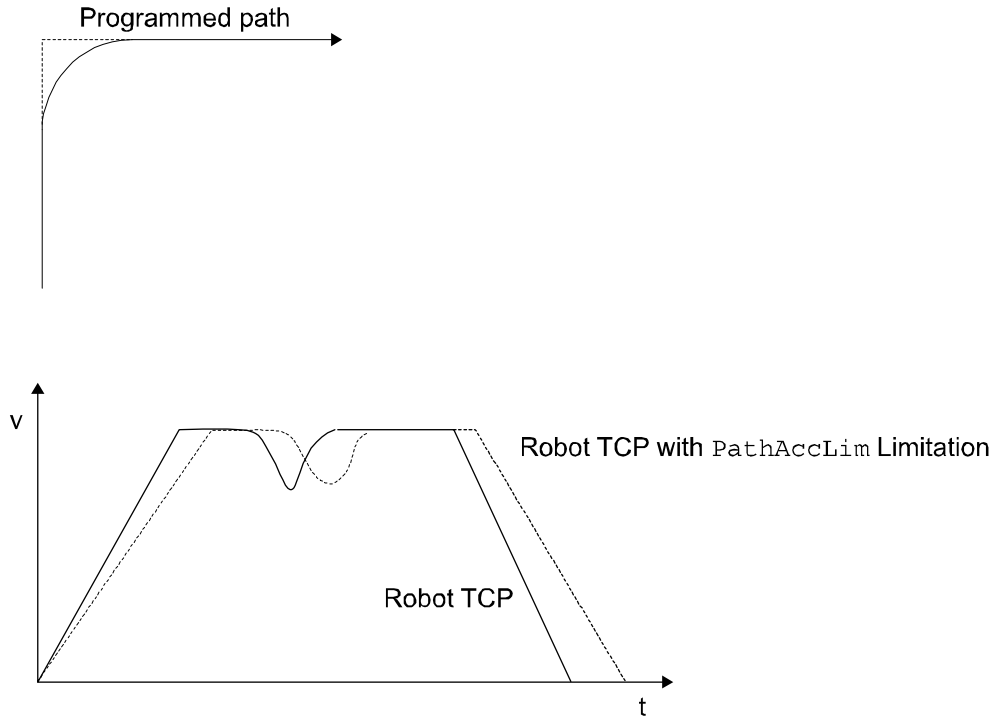
Použití

`PathAccLim` (*Path Acceleration Limitation*) se používá na nastavení nebo resetování omezení na zrychlení a/nebo zpomalení TCP podél dráhy pohybu.

Omezení bude provedeno podél dráhy pohybu, tj. zrychlení v rámci dráhy. Je to tečné zrychlení/zpomalení ve směru dráhy, které bude omezeno.

Instrukce neomezuje celkové zrychlení zařízení, tj. zrychlení ve světovém rámci, takže se nemůže použít přímo k ochraně zařízení před velkými zrychleními.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.



xx0500002184

Základní příklady

Následující příklady názorně ukazují instrukci `PathAccLim`:

Viz také *Další příklady na str 456*.

Příklad 1

```
PathAccLim TRUE \AccMax := 4, TRUE \DecelMax := 4;
```

Zrychlení a zpomalení TCP jsou omezena na 4 m/s^2.

Příklad 2

```
PathAccLim FALSE, FALSE;
```

Zrychlení a zpomalení TCP jsou resetována na maximum (výchozí nastavení).

Argumenty

```
PathAccLim AccLim [\AccMax] DecelLim [\DecelMax]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`AccLim`

Datový typ: `bool`

TRUE, jestliže se chystá omezení zrychlení, jinak FALSE.

`[ \AccMax ]`

Datový typ: `num`

Absolutní hodnota omezení zrychlení v m/sek.^{2}. Použije se pouze když `AccLim` **je TRUE.**

`DecelLim`

Datový typ: `bool`

TRUE, jestliže se chystá omezení zpomalení, jinak FALSE.

`[ \DecelMax ]`

Datový typ: `num`

Absolutní hodnota omezení zpomalení v m/sek.^{2}. Použije se pouze když `DecelLim` **je TRUE.**

Vykonávání programu

Omezení zrychlení/zpomalení se vztahuje k dalšímu provedenému pohybu robotu a je platné až do provedení instrukce `PathAccLim`**.**

Max zrychlení/zpomalení (`PathAccLim FALSE, FALSE`**) se nastavují automaticky**

- **při používání restartovacího režimu Reset RAPID**
- **při načítání nového programu nebo nového modulu**
- **při spuštění vykonávání programu od začátku**
- **při posunu ukazatele programu k** `main`
- **při posunu ukazatele programu k rutině**
- **při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.**

Jestliže existuje kombinace instrukcí `AccSet` **a** `PathAccLim`**, systém omezuje zrychlení/zpomalení v následujícím pořadí:**

- **podle** `AccSet`
- **podle** `PathAccLim`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

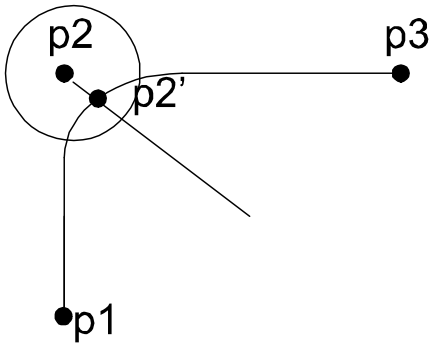
1.159 PathAccLim - Omezit TCP zrychlení podél dráhy

RobotWare - OS

Pokračování

Další příklady

Více příkladů jak používat instrukci `PathAccLim` je názorně uvedeno dole.



xx0500002183

Příklad 1

```
MoveL p1, v1000, fine, tool0;
PathAccLim TRUE\AccMax := 4, FALSE;
MoveL p2, v1000, z30, tool0;
MoveL p3, v1000, fine, tool0;
PathAccLim FALSE, FALSE;
```

TCP zrychlení je omezeno na `4` m/s^2 mezi `p1` a `p3`.

Příklad 2

```
MoveL p1, v1000, fine, tool0;
MoveL p2, v1000, z30, tool0;
PathAccLim TRUE\AccMax :=3, TRUE\DecelMax := 4;
MoveL p3, v1000, fine, tool0;
PathAccLim FALSE, FALSE;
```

TCP zrychlení je omezeno na `3` m/s^2 mezi `p2'` a `p3`.

TCP zpomalení je omezeno na `4` m/s^2 mezi `p2'` a `p3`.

Řešení chyb

Jestliže parametry `\AccMax` nebo `\DecelMax` jsou nastaveny na příliš nízkou hodnotu, systémová proměnná `ERRNO` je nastavena na `ERR_ACC_TOO_LOW`. Tato chyba může být potom ošetřena v chybovém handleru.

Omezení

Min. přípustné zrychlení/zpomalení je `0.1` m/s^2. Doporučujeme mít limit zrychlení a zpomalení symetrický, tj. mít stejnou hodnotu na `AccMax` and `DecelMax`.

Syntaxe

```
PathAccLim
  [ AccLim ':=' ] < expression (IN) of bool >
  [ '\' AccMax ':=' <expression (IN) of num >] ','
  [ DecelLim ':=' ] < expression (IN) of bool>
  [ '\' DecelMax ':=' <expression (IN) of num >] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*
Snížení zrychlení	*AccSet - Omezuje zrychlení na str 19*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.160 PathRecMoveBwd - Posunout záznamník dráhy dozadu

Použití

`PathRecMoveBwd` se používá pro posunutí robotu zpět podél zaznamenané dráhy.

Základní příklady

Následující příklad názorně ukazuje instrukci `PathRecMoveBwd`:

Viz také *Další příklady na str 459*.

Příklad 1

```
VAR pathrecid fixture_id;
PathRecMoveBwd \ID:=fixture_id \ToolOffs:=[0, 0, 10] \Speed:=v500;
```

Robot je posunován zpět k pozici v programu, kde instrukce `PathRecStart` umístila identifikátor `fixture_id`. Ofset TCP je 10 mm ve směru Z a rychlost je nastavena na 500 mm/s.

Argumenty

```
PathRecMoveBwd [\ID] [\ToolOffs] [\Speed]
```

`[\ID]`

Identifier

Datový typ: `pathrecid`

Proměnná, která určuje ID pozici, ke které je třeba se vrátit pozadu. Datový typ `pathrecid` je nehodnotový typ, použitý pouze jako identifikátor pro pojmenování záznamové pozice.

Jestliže není určena žádná ID pozice, je zpětný pohyb v jednoduchém systému prováděn k nejbližší zaznamenané pozici. Ale v synchronizovaném režimu MultiMove se zpětné pohyby provádějí k nejbližší z následujících pozic:

- Zpět k pozici, kde začal synchronizovaný pohyb
- Zpět k nejbližší zaznamenané ID pozici

`[\ToolOffs]`

Tool Offset

Datový typ: `pos`

Stanoví ofset vůle pro TCP během pohybu. Kartézská souřadnice ofsetu se vztahuje k souřadnicím TCP. Kladná hodnota Z ofsetu indikuje vůli. To je vhodné, když robot provádí proces přidávání materiálu. Při provádění synchronizovaného pohybu potom všechny nebo žádná mechanická jednotka potřebují použít tento argument. Jestliže pro některé mechanické jednotky není žádoucí žádný ofset, potom se může použít nulový ofset. I mechanické jednotky bez TCP potřebují používat argument, jestliže se používá TCP robot v odlišné úloze.

`[\Speed]`

Datový typ: `speeddata`

Rychlost nahrazuje rychlost původně použitou během dopředného pohybu. Spedddata definuje rychlost pro středový bod nástroje, reorientaci nástroje a externí

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

osy. Tato rychlost bude použita během zpětného pohybu. Jestliže je vypuštěna, zpětný pohyb se provede rychlostí v původních pohybových instrukcích.

Vykonávání programu

Záznamník dráhy se aktivuje instrukcí `PathRecStart`. Po spuštění záznamníku budou všechny pohybové instrukce zaznamenány a je možné pohybovat robotem zpět podél jeho zaznamenané dráhy u jakéhokoliv bodu vykonáním `PathRecMoveBwd`.

Synchronizovaný pohyb

Provozování záznamníku dráhy v synchronizovaném pohybu nabízí několik úvah.

- Všechny úlohy zúčastněné v zaznamenaném pohybu synchronizace musí přikázat `PathRecMoveBwd`, než se kterýkoliv robot začne pohybovat.
- Veškerá činnost synchronizace je zaznamenána a vykonána obráceně (reverzně). Například, jestliže `PathRecMoveBwd` je přikázán ze synchronizačního bloku k nezávislé pozici, potom záznamník dráhy automaticky změní stav na nezávislý u instrukce `SyncMoveOn`.
- `SyncMoveOn` se považuje za místo přerušení bez identifikátoru dráhy. To znamená, jestliže záznamník dráhy byl spuštěn prostřednictvím `PathRecStart` a `PathRecMoveBwd` bez volitelného argumentu, `\ID` je vykonáno v rámci bloku synchronizovaného pohybu, potom se robot bude pohybovat zpět k pozici, kde robot byl při vykonávání `SyncMoveOn`. Jelikož zpětný pohyb se zastaví před `SyncMoveOn`, stav bude změněn na nezávislý.
- `WaitSyncTask` se považuje za místo přerušení bez identifikátoru dráhy. To znamená, jestliže záznamník dráhy byl spuštěn prostřednictvím `PathRecStart` a `PathRecMoveBwd` je vykonán, potom se robot posune zpět, ne dále než na pozici, kde robot byl, když byl vykonáván `WaitSyncTask`.

Další příklady

Více příkladů jak používat instrukci `PathRecMoveBwd` je názorně uvedeno dole.

Příklad 1 - Nezávislý pohyb

```
VAR pathrecid safe_id;
CONST robtarget p0 := [...];
...
CONST robtarget p4 := [...];
VAR num choice;

MoveJ p0, vmax, z50, tool1;
PathRecStart safe_id;
MoveJ p1, vmax, z50, tool1;
MoveL p2, vmax, z50, tool1;
MoveL p3, vmax, z50, tool1;
MoveL p4, vmax, z50, tool1;

ERROR:
```

Pokračování na další straně

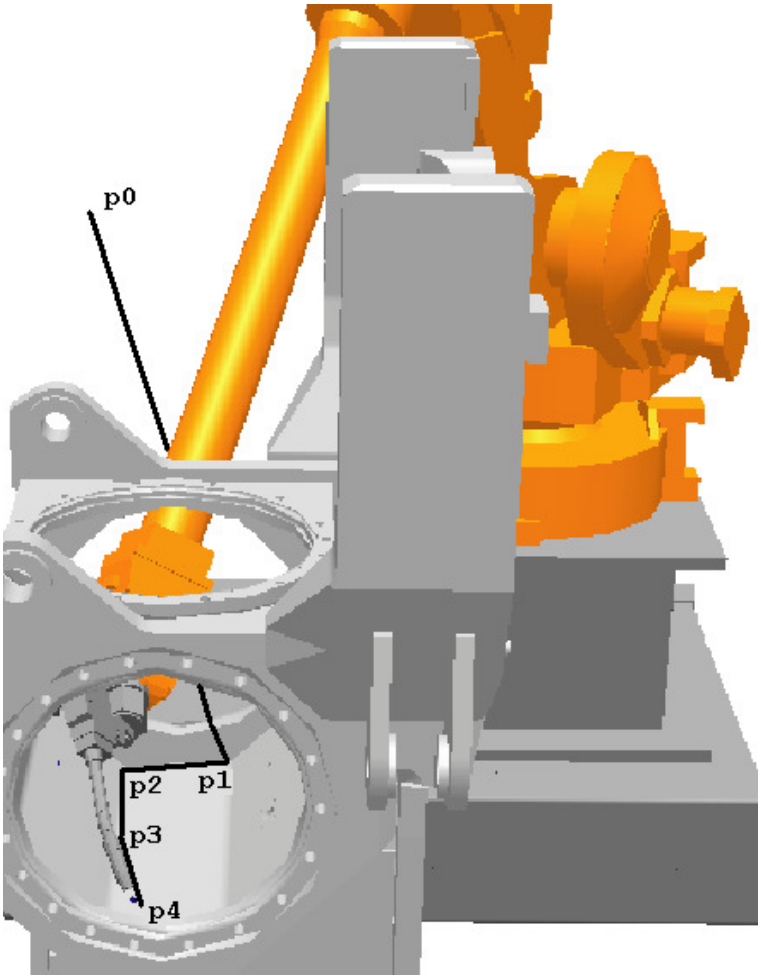
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.160 PathRecMoveBwd - Posunout záznamník dráhy dozadu

Path Recovery

Pokračování

```
TPReadFK choice,"Go to safe?",stEmpty,stEmpty,stEmpty,stEmpty,"Yes";
  IF choice=5 THEN
    IF PathRecValidBwd(\ID:=safe_id) THEN
      StorePath;
      PathRecMoveBwd \ID:=safe_id \ToolOffs:=[0, 0 , 10];
      Stop;
      !Fix problem
      PathRecMoveFwd;
      RestoPath;
      StartMove;
      RETRY;
    ENDIF
  ENDIF
```



xx0500002135

Tento příklad ukazuje, jak může být použit záznamník dráhy pro vytažení robotu z úzkých prostorů po chybě bez programování stanovené dráhy.

Obrobek je vyráběn. V bodě přiblížení `p0` je spuštěn záznamník dráhy a je mu přidělen identifikátor záznamníku dráhy `safe_id`. Představte si, že když se robot pohybuje od `p3` k `p4`, vyskytne se odstranitelná chyba. V tomto bodě je dráha uložena vykonáním `StorePath`. Uložením dráhy může chybový handler začít nový pohyb a později restartovat původní pohyb. Když dráha byla uložena, záznamník

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

dráhy se použije k posunutí robotu ven do bezpečné pozice `p0` vykonáním `PathRecMoveBwd`.

Všimněte si, že ofset nástroje se použije k vytvoření vůle od, například, nově přidaného svaru. Když byl robot posunut ven, operátor může udělat, co je nezbytné, aby opravil chybu (například vyčistit svařovací hořák). Potom je robot posunut zpět na místo chyby pomocí `PathRecMoveFwd`. Na místě chyby je úroveň dráhy přepnuta zpět na úroveň základny pomocí `RestoPath` a je proveden nový pokus.

Příklad 2 - Synchronizovaný pohyb

T_ROB1

```
VAR pathrecid HomeROB1;
CONST robtarget pR1_10:=[...];
...
CONST robtarget pR1_60:=[...];

PathRecStart HomeROB1;
MoveJ pR1_10, v1000, z50, tGun;
MoveJ pR1_20, v1000, z50, tGun;
MoveJ pR1_30, v1000, z50, tGun;
SyncMoveOn sync1, tasklist;
MoveL pR1_40 \ID:=1, v1000, z50, tGun\wobj:=pos1;
MoveL pR1_50 \ID:=2, v1000, z50, tGun\wobj:=pos1;
MoveL pR1_60 \ID:=3, v1000, z50, tGun\wobj:=pos1;
SyncMoveOff sync2;

ERROR
  StorePath \KeepSync;
  TEST ERRNO
  CASE ERR_PATH_STOP:
    PathRecMoveBwd \ID:= HomeROB1\ToolOffs:=[0,0,10];
  ENDTEST
  !Perform service action
  PathRecMoveFwd \ToolOffs:=[0,0,10];
  RestoPath;
  StartMove;
```

T_ROB2

```
VAR pathrecid HomeROB2;
CONST robtarget pR2_10:=[...];
...
CONST robtarget pR2_50:=[...];

PathRecStart HomeROB2;
MoveJ pR2_10, v1000, z50, tGun;
MoveJ pR2_20, v1000, z50, tGun;
SyncMoveOn sync1, tasklist;
MoveL pR2_30 \ID:=1, v1000, z50, tGun\wobj:=pos1;
MoveL pR2_40 \ID:=2, v1000, z50, tGun\wobj:=pos1;
MoveL pR2_50 \ID:=3, v1000, z50, tGun\wobj:=pos1;
SyncMoveOff sync2;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.160 PathRecMoveBwd - Posunout záznamník dráhy dozadu

Path Recovery

Pokračování

```
ERROR
  StorePath \KeepSync;
  TEST ERRNO
  CASE ERR_PATH_STOP:
    PathRecMoveBwd \ToolOffs:=[0,0,10];
  ENDTEST
  !Perform service action
  PathRecMoveFwd \ToolOffs:=[0,0,10];
  RestoPath;
  StartMove;
```

T_ROB3

```
VAR pathrecid HomePOS1;
CONST jointtarget jP1_10:=[...];
...
CONST jointtarget jP1_40:=[...];

PathRecStart HomePOS1;
MoveExtJ jP1_10, v1000, z50;
SyncMoveOn sync1, tasklist;
MoveExtJ jP1_20 \ID:=1, v1000, z50;
MoveExtJ jP1_30 \ID:=2, v1000, z50;
MoveExtJ jP1_40 \ID:=3, v1000, z50;
SyncMoveOff sync2;

ERROR
  StorePath \KeepSync;
  TEST ERRNO
  CASE ERR_PATH_STOP:
    PathRecMoveBwd \ToolOffs:=[0,0,0];
  DEFAULT:
  PathRecMoveBwd \ID:=HomePOS1\ToolOffs:=[0,0,0];
  ENDTEST
  !Perform service action
  PathRecMoveFwd \ToolOffs:=[0,0,0];
  RestoPath;
  StartMove;
```

Systém se skládá ze tří manipulátorů, které všechny běží v oddělených úlohách. Předpokládejte, že `T_ROB1` **zaznamená chybu** `ERR_PATH_STOP` **v synchronizovaném bloku** `sync1`**. Po chybě je nutné přejít zpět do výchozí (home) pozice označené identifikátorem záznamníku dráhy** `HomeROB1` **kvůli provedení servisu externího vybavení robotu. To se provádí pomocí** `PathRecMoveBwd` **a dodáním identifikátoru** `pathrecid`**.**

Jelikož chyba vznikla během synchronizovaného pohybu, je nutné, aby druhý TCP robot `T_ROB2` **a externí osa** `T_POS1` **také přikázaly** `PathRecMoveBwd`**. Tyto manipulátory se nemusejí vracet zpět dále než kde synchronizovaný pohyb začal. Nedodáním** `PathRecMoveBwd` **na** `ERR_PATH_STOP` **s identifikátorem záznamníku dráhy se využije schopnost záznamníku dráhy zastavit po** `SyncMoveOn`**. Všimněte si, že externí osa, která nemá TCP, stále dodává nulový ofset nástroje, aby byla aktivována schopnost TCP robotů to provést.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

DEFAULT chování v ERROR handleru v tomto příkladu je takové, že všechny manipulátory nejprve provádějí synchronizované pohyby zpět a potom nezávislé pohyby zpět k bodu startu zaznamenané dráhy. Toho je dosaženo určením \ID v PathRecMoveBwd pro všechny manipulátory.

Omezení

Pohyby pomocí záznamníku dráhy nemohou být provedeny na úrovni základny, tzn. StorePath je nutné vykonat před PathRecMoveBwd.

Nikdy není možné pohybovat se zpětně přes příkaz SynchMoveOff.

Nikdy není možné pohybovat se zpětně přes příkaz WaitSyncTask.

SyncMoveOn musí být předcházen alespoň jedním nezávislým pohybem, jestliže se chceme pohybovat zpět k pozici, kde začal synchronizovaný pohyb.

Jestliže není žádoucí vracet se k bodu, kde byl vykonán PathRecMoveBwd (vykonáním PathRecMoveFwd), potom musí být záznamník dráhy zastaven pomocí PathRecStop. PathRecStop\Clear také vyčistí zaznamenanou dráhu.

PathRecMoveBwd se nemůže provádět v RAPID rutině připojené k jakékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart,, Reset nebo Step.

Syntaxe

```
PathRecMoveBwd
  [ '\' ID ':=' < variable (VAR) of pathrecid > ]
  [ '\' ToolOffs ':=' <expression (IN) of pos> ]
  [ '\' Speed ':=' <expression (IN) of speeddata> ]';'
```

Související informace

Pro informace o	Viz
Identifikátor záznamníku dráhy	*pathrecid - Identifikátor záznamníku dráhy na str 1551*
Spustit - zastavit záznamník dráhy	*PathRecStart - Spustit záznamník dráhy na str 467* *PathRecStop - Zastavit záznamník dráhy na str 470*
Zkontrolovat platnou zaznamenanou dráhu	*PathRecValidBwd - Je zaznamenána platná zpětná dráha na str 1255* *PathRecValidFwd - Je zaznamenána platná dráha dopředu na str 1259*
Posunout záznamník dráhy dopředu	*PathRecMoveFwd - Posunout záznamník dráhy dopředu na str 464*
Uložit - obnovit dráhy	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742* *RestoPath - Obnovuje cestu po přerušení na str 546*
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Obnovení po chybě	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.161 PathRecMoveFwd - Posunout záznamník dráhy dopředu

Použití

`PathRecMoveFwd` se používá k posunutí robotu zpět k pozici, kde byl vykonán `PathRecMoveBwd`. Je také možné pohybovat robotem částečně dopředu dodáním identifikátoru, který byl překročen během zpětného pohybu.

Základní příklady

Následující příklad názorně ukazuje instrukci `PathRecMoveFwd`:

Viz také *Další příklady na str 465*.

Příklad 1

```
PathRecMoveFwd;
```

Robot je posunut zpět k pozici, kde záznamník dráhy spustil zpětný pohyb.

Argumenty

```
PathRecMoveFwd [\ID] [\ToolOffs] [\Speed]
```

[\ID]

Identifier

Datový typ: `pathrecid`

Proměnná, která určuje ID pozici, ke které je třeba se posunout dopředu. Datový typ `pathrecid` je nehodnotový typ, použitý pouze jako identifikátor pro pojmenování záznamové pozice.

Jestliže není určena žádná ID pozice, potom bude vždy proveden dopředný pohyb k pozici přerušení na původní dráze.

[\ToolOffs]

Tool Offset

Datový typ: `pos`

Poskytuje ofset vůle pro TCP během pohybu. Kartézská souřadnice je aplikována na souřadnice TCP. To je výhodné, když robot provádí proces přidávání materiálu.

`[\Speed]`

Datový typ: `speeddata`

Rychlost potlačuje rychlost původně použitou během dopředného pohybu. Speeddata definuje rychlost pro středový bod nástroje, reorientaci nástroje a externí osy. Tato rychlost bude použita během zpětného pohybu. Jestliže je vypuštěna, dopředný pohyb se provede rychlostí v původních pohybových instrukcích.

Vykonávání programu

Záznamník dráhy se aktivuje s instrukcí `PathRecStart`. Po spuštění záznamníku může být robot posunut zpětně podél své provedené dráhy vykonáním `PathRecMoveBwd`. Robot může být potom voláním `PathRecMoveFwd` přikázán zpět na pozici, kde začalo zpětné vykonávání. Je také možné posunout robot částečně dopředu dodáním identifikátoru, který byl předán během zpětného pohybu.

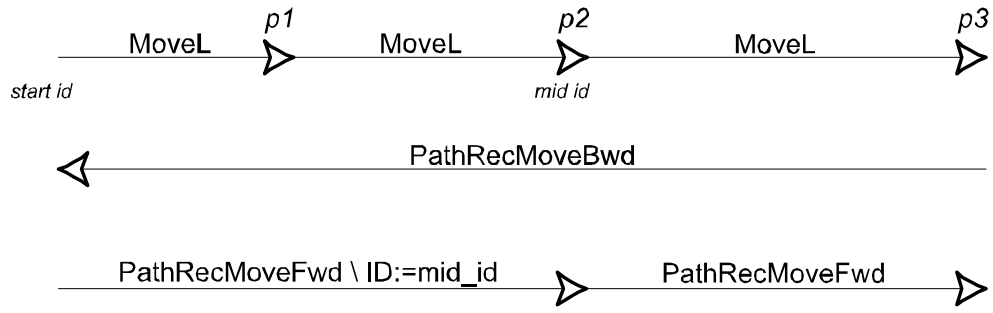
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Více příkladů jak používat instrukci `PathRecMoveFwd` je názorně uvedeno dole.

```
VAR pathrecid start_id;
VAR pathrecid mid_id;
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];

  PathRecStart start_id;
  MoveL p1, vmax, z50, tool1;
  MoveL p2, vmax, z50, tool1;
  PathRecStart mid_id;
  MoveL p3, vmax, z50, tool1;
  StorePath;
  PathRecMoveBwd \ID:=start_id;
  PathRecMoveFwd \ID:=mid_id;
  PathRecMoveFwd;
  RestoPath;
```



xx0500002133

Příklad nahoře spustí záznamník dráhy a bod startu bude označen identifikátorem dráhy `start_id`. Potom se robot bude pohybovat dopředu s tradičními pohybovými instrukcemi a potom se bude pohybovat zpět k identifikátoru záznamníku dráhy `start_id` pomocí zaznamenané dráhy. Nakonec se bude pohybovat znovu dopředu dvěma kroky pomocí `PathRecMoveFwd`.

Omezení

Pohyby pomocí záznamníku dráhy se musí provádět na úrovni trap, tj. `StorePath` se musí vykonat před `PathRecMoveFwd`.

Aby bylo možné vykonat `PathRecMoveFwd`, musí být předtím vykonán `PathRecMoveBwd`.

Jestliže není žádoucí vracet se k bodu, kde byl vykonán `PathRecMoveBwd` (vykonáním `PathRecMoveFwd`), potom musí být záznamník dráhy zastaven pomocí `PathRecStop`. `PathRecStop\Clear` také vyčistí zaznamenanou dráhu.

`PathRecMoveFwd` se nemůže provádět v RAPID rutině připojené k jakékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart, Reset nebo Step.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.161 PathRecMoveFwd - Posunout záznamník dráhy dopředu

PathRecovery

Pokračování

Syntaxe

```
PathRecMoveFwd '('
  [ '\' ID ':=' < variable (VAR) of pathid > ]
  [ '\' ToolOffs ':=' <expression (IN) of pos> ]
  [ '\' Speed ':=' <expression (IN) of speeddata> ]';'
```

Související informace

Pro informace o	Viz
Identifikátory záznamníku dráhy	*pathrecid - Identifikátor záznamníku dráhy na str 1551*
Spustit - zastavit záznamník dráhy	*PathRecStart - Spustit záznamník dráhy na str 467* *PathRecStop - Zastavit záznamník dráhy na str 470*
Zkontrolovat platnou zaznamenanou dráhu	*PathRecValidBwd - Je zaznamenána platná zpětná dráha na str 1255* *PathRecValidFwd - Je zaznamenána platná dráha dopředu na str 1259*
Posunout záznamník dráhy dozadu	*PathRecMoveBwd - Posunout záznamník dráhy dozadu na str 458*
Uložit - obnovit dráhy	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742* *RestoPath - Obnovuje cestu po přerušení na str 546*
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Obnovení po chybě	*Technická referenční příručka - Přehled RAPID* *Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.162 PathRecStart - Spustit záznamník dráhy

Použití

`PathRecStart` se používá pro spuštění záznamu dráhy robotu. Záznamník dráhy uloží informaci dráhy během vykonávání programu RAPID.

Základní příklady

Následující příklad názorně ukazuje instrukci `PathRecStart`:

Příklad 1

```
VAR pathrecid fixture_id;

PathRecStart fixture_id;
```

Záznamník dráhy je spuštěn a bod startu (pozice instrukce v programu RAPID) je označen identifikátorem `fixture_id`.

Argumenty

```
PathRecStart ID
```

`ID`

Identifier

Datový typ:`pathrecid`

Proměnná, která určuje jméno startovní pozice záznamu. Datový typ `pathrecid` je nehodnotový typ, použitý pouze jako identifikátor pro pojmenování záznamové pozice.

Vykonávání programu

Když je záznamník dráhy přikázán ke spuštění, dráha robotu bude vnitřně zaznamenána v řadiči robotu. Zaznamenaná sekvence programových pozic může být přenesena zpětně pomocí `PathRecMoveBwd`, což způsobí zpětný pohyb robotu podél jeho vykonané dráhy.

Další příklady

Více příkladů jak používat instrukci `PathRecStart` je názorně uvedeno dole.

Příklad 1

```
VAR pathrecid origin_id;
VAR pathrecid corner_id;
VAR num choice;
MoveJ p1, vmax, z50, tool1;
PathRecStart origin_id;
MoveJ p2, vmax, z50, tool1;
PathRecStart corner_id;
MoveL p3, vmax, z50, tool1;
MoveAbsJ jt4, vmax, fine, tool1;
ERROR
  TPReadFK choice,"Extract
       to:",stEmpty,stEmpty,stEmpty,"Origin","Corner";
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.162 PathRecStart - Spustit záznamník dráhy

Path Recovery

Pokračování

```
IF choice=4 OR choice=5 THEN
  StorePath;
  IF choice=4 THEN
    PathRecMoveBwd \ID:=origin_id;
  ELSE
    PathRecMoveBwd \ID:=corner_id;
  ENDIF
  Stop;
  !Fix problem
  PathRecMoveFwd;
  RestoPath;
  StartMove;
  RETRY;
ENDIF
```

V příkladu nahoře je záznamník dráhy použit k pohybu robotu k servisní pozici, jestliže vznikne chyba během normálního vykonávání.

Robot je veden podél dráhy. Za pozicí `p1` je spuštěn záznamník dráhy. Za bodem `p2` je vložen další identifikátor dráhy. Předpokládejte, že se objeví odstranitelná chyba při pohybu od pozice `p3` k pozici `jt4`. Chybový handler bude nyní vyvolán a uživatel může zvolit mezi vytažením robotu na pozici `Origin` (bod `p1`) nebo `Corner` (bod `p2`). Potom je úroveň dráhy přepnuta s `StorePath`, aby bylo možné později znovu začít v místě chyby. Když je robot vrácen od místa chyby, je na uživateli, aby chybu vyřešil (obvykle opravou okolního vybavení robotu).

Potom je robot přikázán zpět k místu chyby. Úroveň dráhy je přepnuta zpět na normální a je proveden nový pokus.

Omezení

Záznamník dráhy může být pouze spuštěn a bude pouze zaznamenávat dráhu v úrovni základny, tj. pohyby na úrovni `StorePath` nejsou zaznamenány.

Syntaxe

```
PathRecStart
  [ ID ':='] < variable (VAR) of pathrecid> ';'
```

Související informace

Pro informace o	Viz
Identifikátory záznamníku dráhy	*pathrecid - Identifikátor záznamníku dráhy na str 1551*
Zastavit záznamník dráhy	*PathRecStop - Zastavit záznamník dráhy na str 470*
Zkontrolovat platnou zaznamenanou dráhu	*PathRecValidBwd - Je zaznamenána platná zpětná dráha na str 1255* *PathRecValidFwd - Je zaznamenána platná dráha dopředu na str 1259*
Přehrát záznamník dráhy dozadu	*PathRecMoveBwd - Posunout záznamník dráhy dozadu na str 458*
Přehrát záznamník dráhy dopředu	*PathRecMoveFwd - Posunout záznamník dráhy dopředu na str 464*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.163 PathRecStop - Zastavit záznamník dráhy

Použití

`PathRecStop`se používá pro zastavení záznamu dráhy robotu.

Základní příklady

Následující příklad názorně ukazuje instrukci `PathRecStop`:

Viz také *Další příklady* dole.

Příklad 1

```
PathRecStop \Clear;
```

Záznamník dráhy je zastaven a zásobník uložených informací o dráze je vyčištěn.

Argumenty

```
PathRecStop [\Clear]
```

`[\Clear]`

Datový typ: `switch`

Vyčistit zaznamenanou dráhu.

Vykonávání programu

Když je záznamník dráhy přikázán k zastavení, záznam dráhy se zastaví. Volitelný argument `\Clear` vyčistí zásobník uložených informací o dráze, což zabrání chybnému vykonání zaznamenané dráhy.

Po zastavení záznamníku s `PathRecStop` se dříve zaznamenané dráhy nemohou použít pro zpětné pohyby (`PathRecMoveBwd`). Je možné použít dříve zaznamenané dráhy, jestliže je znovu přikázán `PathRecStart` od stejné pozice, ve které byl záznamník dráhy zastaven. Viz následující příklad.

Další příklady

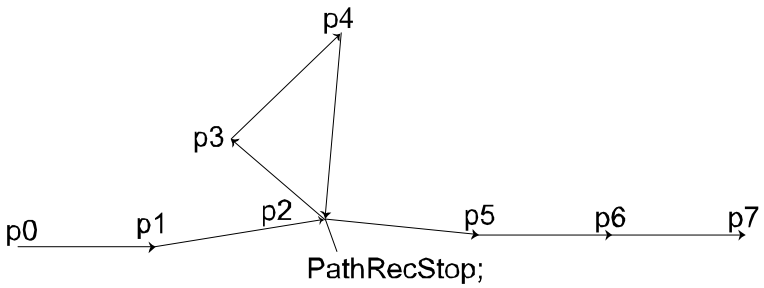
Více příkladů jak používat instrukci `PathRecStop` je názorně uvedeno dole.

```
LOCAL VAR pathrecid id1;
LOCAL VAR pathrecid id2;
LOCAL CONST robtarget p0:= [...];
......
LOCAL CONST robtarget p6 := [...];
PROC example1()
  MoveL p0, vmax, z50, tool1;
  PathRecStart id1;
  MoveL p1, vmax, z50, tool1;
  MoveL p2, vmax, z50, tool1;
  PathRecStop;
  MoveL p3, vmax, z50, tool1;
  MoveL p4, vmax, z50, tool1;
  MoveL p2, vmax, z50, tool1;
  PathRecStart id2;
  MoveL p5, vmax, z50, tool1;
  MoveL p6, vmax, z50, tool1;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  StorePath;
  PathRecMoveBwd \ID:=id1;
  PathRecMoveFwd;
  RestoPath;
  StartMove;
  MoveL p7, vmax, z50, tool1;
ENDPROC
PROC example2()
  MoveL p0, vmax, z50, tool1;
  PathRecStart id1;
  MoveL p1, vmax, z50, tool1;
  MoveL p2, vmax, z50, tool1;
  PathRecStop;
  MoveL p3, vmax, z50, tool1;
  MoveL p4, vmax, z50, tool1;
  PathRecStart id2;
  MoveL p2, vmax, z50, tool1;
  MoveL p5, vmax, z50, tool1;
  MoveL p6, vmax, z50, tool1;
  StorePath;
  PathRecMoveBwd \ID:=id1;
  PathRecMoveFwd;
  RestoPath;
  StartMove;
  MoveL p7, vmax, z50, tool1;
ENDPROC
```



PathRecStop_

Příklady nahoře popisují zaznamenávání dráhy robotu, když je záznam zastaven uprostřed sekvence. V `example1` **je příkaz** `PathRecMoveBwd \ID:=id1;` **platný a robot bude vykonávat následující dráhu:** `p6 -> p5 -> p2 -> p1 -> p0`

Důvod, proč je příkaz platný: záznamník byl zastaven a spuštěn v přesně stejné pozici robotu. Jestliže toto chování je nežádoucí, stop příkaz by měl zahrnovat volitelný argument `\Clear`**. Tím pádem bude zaznamenaná dráha vyčištěna a nikdy už nebude možné vrátit se k předchozím identifikátorům záznamníku dráhy.**

Jediným rozdílem v `example2` **je to, kde byl záznamník spuštěn podruhé. V tomto případě** `PathRecMoveBwd \ID:=id1` **způsobí chybu. To je proto, že neexistuje žádná zaznamenaná dráha mezi** `p4`**,** `p3` **a** `p2`**. Je možné vykonat** `PathRecMoveBwd \ID:=id2`**.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.163 PathRecStop - Zastavit záznamník dráhy

Path Recovery

Pokračování

Syntaxe

```
PathRecStop
  [ '\'switch Clear ] ';'
```

Související informace

Pro informace o	Viz
Identifikátory záznamníku dráhy	*pathrecid - Identifikátor záznamníku dráhy na str 1551*
Spustit záznamník dráhy	*PathRecStart - Spustit záznamník dráhy na str 467*
Zkontrolovat platnou zaznamenanou dráhu	*PathRecValidBwd - Je zaznamenána platná zpětná dráha na str 1255* *PathRecValidFwd - Je zaznamenána platná dráha dopředu na str 1259*
Přehrát záznamník zpětně	*PathRecMoveBwd - Posunout záznamník dráhy dozadu na str 458*
Přehrát záznamník dopředu	*PathRecMoveFwd - Posunout záznamník dráhy dopředu na str 464*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.164 PathResol - Potlačit rozlišení dráhy

Použití

`PathResol` (*Path Resolution*) se používá k potlačení (override) času konfigurovaného vzorku geometrické dráhy v systémových parametrech pro mechanické jednotky, které jsou řízené aktuální programovou úlohou.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* ve všech pohybových úlohách.

Popis

Rozlišení dráhy ovlivňuje přesnost interpolované dráhy a dobu trvání programového cyklu. Přesnost dráhy se zlepší a doba cyklu se často zkrátí, když je snížen parametr `PathSampleTime`. Hodnota pro parametr `PathSampleTime`, která je příliš nízká, může způsobit problémy se zátěží CPU v některých náročných aplikacích. Používání standardního konfigurovaného rozlišení dráhy (`PathSampleTime` 100%) odstraní problémy CPU se zatížením a poskytne dostatečnou přesnost dráhy ve většině situací.

Příklad použití `PathResol`:

Dynamicky kritické pohyby (max užitečná zátěž, vysoká rychlost, kombinované pohyby spoje blízko hranice pracovní oblasti) mohou způsobit problémy s přetížením CPU. Zvyšte parametr `PathSampleTime`.

Externí osy s nízkým výkonem mohou způsobit problémy s přetížením CPU během koordinace. Zvyšte parametr `PathSampleTime`.

Obloukové svařování s vysokofrekvenčním weavingem může vyžadovat vysoké rozlišení interpolované dráhy. Snižte parametr `PathSampleTime`.

Malé kruhy nebo malé pohyby se změnami směru mohou snížit kvalitu činnosti dráhy a prodloužit dobu cyklu. Snižte parametr `PathSampleTime`.

Lepení s velkými reorientacemi a malými rohovými zónami mohou způsobit kolísání rychlosti. Snižte parametr `PathSampleTime`.

Základní příklady

Následující příklad názorně ukazuje instrukci `PathResol`:

```
MoveJ p1,v1000,fine,tool1;
PathResol 150;
```

S robotem na stop bodu se vzorkový čas dráhy zvyšuje na `150` % konfigurovaného.

Argumenty

```
PathResol PathSampleTime
```

`PathSampleTime`

Datový typ: `num`

Potlačení jako procento času vzorku konfigurované dráhy. 100 % odpovídá času vzorku konfigurované dráhy. V rámci rozsahu 25-400 %.

Nižší hodnota parametru `PathSampleTime` zlepšuje rozlišení dráhy (přesnost dráhy).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.164 PathResol - Potlačit rozlišení dráhy

RobotWare - OS

Pokračování

Vykonávání programu

Rozlišení dráhy všech následných polohovacích instrukcí jsou ovlivněna až do vykonání nové instrukce `PathResol`. To ovlivní rozlišení dráhy během vykonávání pohybů všech programů (výchozí úroveň dráhy a úroveň dráhy po `StorePath`) a také během ručního posunu (jogging).

V systému MultiMove v synchronizovaném koordinovaném režimu jsou platné následující body:

- Všechny mechanické jednotky zúčastněné v synchronizovaném koordinovaném režimu poběží s aktuálním rozlišením dráhy pro aktuální (používaný) plánovač pohybů.
- Nové pořadí rozlišení dráhy proti aktuálnímu plánovači pohybů ovlivní synchronizovaný koordinovaný pohyb a budoucí nezávislý pohyb v tomto plánovači pohybů.
- Nové pořadí rozlišení dráhy proti jinému plánovači pohybů ovlivní pouze nezávislý pohyb v tomto plánovači pohybů.

Podívejte se na informace o spojení mezi programovou úlohou a plánovačem pohybů *Aplikační příručka - MultiMove*.

Výchozí hodnota pro potlačení doby vzorku dráhy je 100 %. Tato hodnota se nastavuje automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Aktuální potlačení času vzorku dráhy je možné přečíst z proměnné `C_MOTSET` (datový typ `motsetdata`) v komponentu `pathresol`.

Omezení

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (zonedata `fine`), nikoliv s fly-by bodem, jinak nebude možný restart po selhání napájení.

`PathResol` se nemůže provádět v RAPID rutině připojené ke kterékoliv z následujících speciálních systémových událostí: PowerOn, Stop, Restart, QStop, nebo Step.

Syntaxe

```
PathResol
  [PathSampleTime ':=' ] < expression (IN) of num>' ;'
```

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Nastavení pohybu	*Technická referenční příručka - Přehled RAPID*
Konfigurace rozlišení dráhy	*Technická referenční příručka - Systémové parametry*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.165 PDispOff - Deaktivuje posun programu

Použití

`PDispOff` (*Program Displacement Off*) se používá pro deaktivaci posunu programu.

Posunutí programu se aktivuje instrukcí `PDispSet` nebo `PDispOn` a vztahuje se na všechny pohyby až do aktivace některého jiného posunu programu nebo do deaktivace posunu programu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `PDispOff`:

Příklad 1

```
PDispOff;
```

Deaktivace posunu programu.

Příklad 2

```
MoveL p10, v500, z10, tool1;
PDispOn \ExeP:=p10, p11, tool1;
MoveL p20, v500, z10, tool1;
MoveL p30, v500, z10, tool1;
PDispOff;
MoveL p40, v500, z10, tool1;
```

Posun programu je definován jako rozdíl mezi pozicemi `p10` a `p11`. Tento posun ovlivňuje pohyb k `p20` a `p30`, ale nikoliv k `p40`.

Vykonávání programu

Aktivní posun programu je resetován. To znamená, že souřadný systém posunu programu je stejný jako souřadný systém objektu a tedy všechny naprogramované pozice budou souviset s posledně uvedeným.

Syntaxe

```
PDispOff ´;'
```

Související informace

Pro informace o	Viz
Definice posunu programu pomocí dvou pozic	*PDispOn - Aktivuje posun programu na str 477*
Definice posunu programu pomocí známého rámce	*PDispSet - Aktivuje posun programu pomocí známého rámce na str 481*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.166 PDispOn - Aktivuje posun programu

Použití

`PDispOn` (*Program Displacement On*) se používá k definování a aktivaci posunu programu pomocí dvou pozic robotu.

Posun programu se používá například po provedení hledání nebo když podobné vzorce pohybu jsou opakovány na několika různých místech programu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `PDispOn`:

Viz také *Další příklady na str 479*.

Příklad 1

```
MoveL p10, v500, z10, tool1;
PDispOn \ExeP:=p10, p20, tool1;
```

Aktivace posunu programu (paralelní posun). Vypočítá se na základě rozdílu mezi pozicemi `p10` a `p20`.

Příklad 2

```
MoveL p10, v500, fine \Inpos := inpos50, tool1;
PDispOn *, tool1;
```

Aktivace posunu programu (paralelní posun). Jelikož stop bod, který je přesně definován, byl použit v předchozí instrukci, argument `\ExeP` není nutné používat. Posun je vypočítán na základě rozdílu mezi aktuální pozicí robotu a naprogramovaným bodem (*) uloženým v instrukci.

Příklad 3

```
PDispOn \Rot \ExeP:=p10, p20, tool1;
```

Aktivace posunu programu včetně rotace. Vypočítá se na základě rozdílu mezi pozicemi `p10` a `p20`.

Argumenty

```
PDispOn [\Rot] [\ExeP] ProgPoint Tool [\WObj]
```

`[ \Rot ]`

Rotation

Datový typ: `switch`

Rozdíl v orientaci nástroje se bere v úvahu a toto představuje rotaci v programu.

`[ \ExeP ]`

Executed Point

Datový typ: `robtarget`

Nová pozice robotu použitá pro výpočet posunu. Jestliže je tento argument vypuštěn, použije se aktuální pozice robotu v době vykonávání programu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.166 PDispOn - Aktivuje posun programu

RobotWare - OS

Pokračování

`ProgPoint`

Programmed Point

Datový typ: `robtarget`

Původní pozice robotu v době programování.

`Tool`

Datový typ: `tooldata`

Nástroj použitý během programování, tj. TCP, ke kterému je vztažena pozice `ProgPoint`.

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

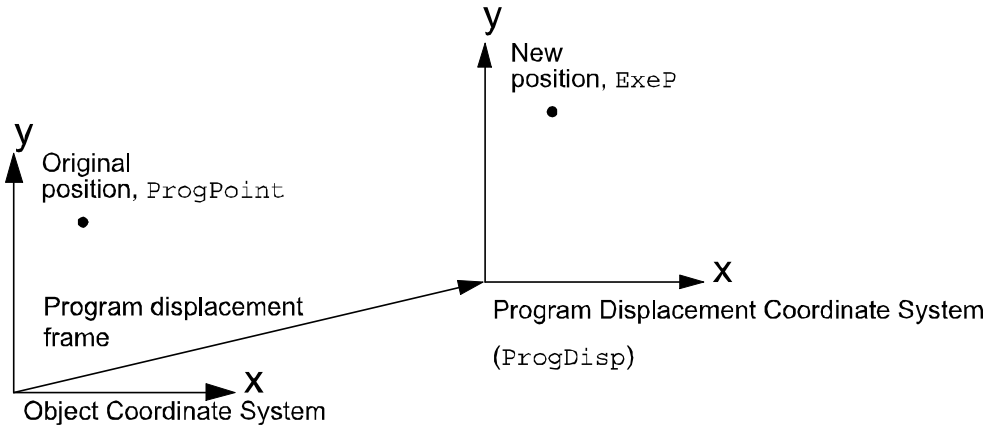
Pracovní objekt (souřadný systém), ke kterému se vztahuje pozice `ProgPoint`.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže je použit stacionární TCP nebo koordinované externí osy, tento argument musí být potom stanoven.

Argumenty Tool a \WObj se používají pro výpočet `ProgPoint` **během programování a pro výpočet aktuální pozice během vykonávání programu, jestliže není naprogramován žádný argument** `\ExeP`.

Vykonávání programu

Posun programu znamená, že souřadný systém `ProgDisp` **je přeložen ve vztahu k souřadnému systému objektu. Jelikož všechny pozice jsou vztaženy k souřadnému systému** `ProgDisp`**, všechny naprogramované pozice budou také posunuty. Viz obrázek dole, který ukazuje paralelní posun naprogramované pozice pomocí posunu programu.**



xx0500002186

Posun programu se aktivuje, když je vykonána instrukce `PDispOn` **a zůstává aktivní až do doby aktivace jiného posunu programu (instrukce** `PDispSet` **nebo** `PDispOn`**) nebo do deaktivace posunu programu (instrukce** `PDispOff`**).**

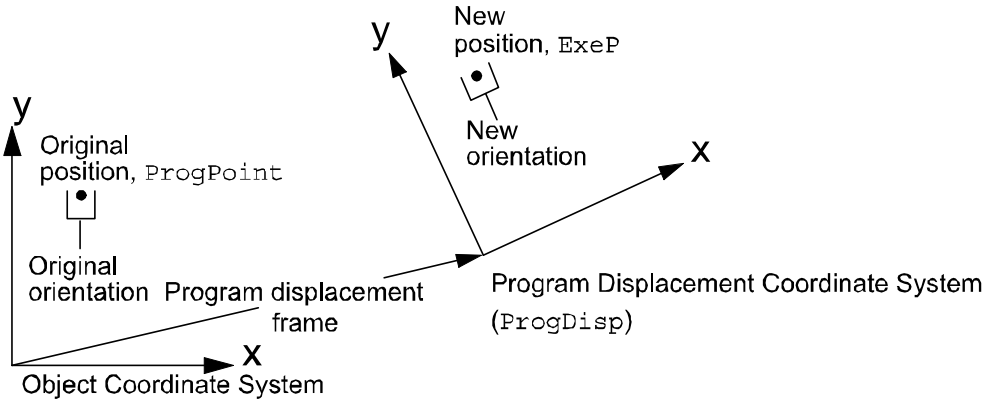
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pouze jeden posun programu může být aktivní ve stejnou dobu. Na druhé straně, několik instrukcí `PDispOn` může být naprogramováno jedna po druhé, a v tomto případě budou přidány odlišné posuny programu.

Posun programu se vypočítává jako rozdíl mezi `ExeP` a `ProgPoint`. Jestliže nebylo určeno `ExeP`, použije se místo toho aktuální pozice robotu v době vykonávání programu. Jelikož se používá aktuální pozice robotu, robot by se neměl při vykonávání `PDispOn` pohybovat.

Jestliže je použit argument `\Rot`, potom se také rotace vypočítává na základě orientace nástroje na dvou pozicích. Posun bude vypočítán takovým způsobem, že nová pozice (`ExeP`) bude mít stejnou pozici a orientaci ve vztahu posunutému souřadnému systému `ProgDisp` jako měla stará pozice (`ProgPoint`) ve vztahu k původnímu souřadnému systému objektu. Viz obrázek dole, který ukazuje překlad a rotaci naprogramované pozice.



xx0500002187

Posun programu se nastavuje automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Další příklady

Více příkladů jak používat instrukci `PDispOn` je názorně uvedeno dole.

Příklad 1

```
PROC draw_square()
  PDispOn *, tool1;
  MoveL *, v500, z10, tool1;
  MoveL *, v500, z10, tool1;
  MoveL *, v500, z10, tool1;
  MoveL *, v500, z10, tool1;
  PDispOff;
ENDPROC
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

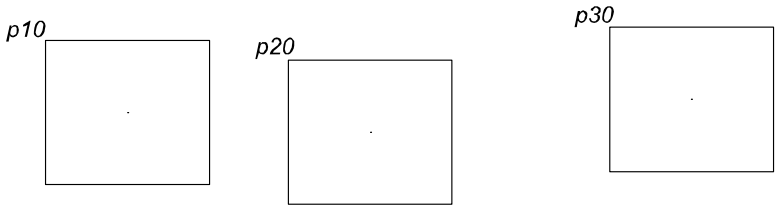
1.166 PDispOn - Aktivuje posun programu

RobotWare - OS

Pokračování

```
...
MoveL p10, v500, fine \Inpos := inpos50, tool1;
draw_square;
MoveL p20, v500, fine \Inpos := inpos50, tool1;
draw_square;
MoveL p30, v500, fine \Inpos := inpos50, tool1;
draw_square;
```

Rutina `draw_square` se používá k vykonání stejného vzorce pohybu na třech různých pozicích na základě pozic `p10`, `p20`, a `p30`. Viz obrázek dole, který ukazuje, že při používání posunu programu mohou být znovu použity vzorce pohybu.



xx0500002185

Příklad 2

```
SearchL sen1, psearch, p10, v100, tool1\WObj:=fixture1;
PDispOn \ExeP:=psearch, *, tool1 \WObj:=fixture1;
```

Je provedeno hledání, při kterém je nalezená pozice robotu uložena do pozice `psearch`. Každý pohyb provedený potom začíná od této pozice pomocí posunu programu (paralelní posun). Naposledy zmíněný je vypočítán na základě rozdílu mezi hledanou pozicí a naprogramovaným bodem (`*`) uloženým v instrukci. Všechny pozice jsou založeny na souřadném systému objektu `fixture1`.

Syntaxe

```
PDispOn
  [ [ ’\’ Rot ]
    [’\’ ExeP ’:=’ < expression (IN) of robtarget>]’,’]
  [ ProgPoint’ :=’ ] < expression (IN) of robtarget> ’,’
  [ Tool ’:=’ ] < persistent (PERS) of tooldata>
  [ ´\’WObj’ :=’ < persistent (PERS) of wobjdata> ] ´;’
```

Související informace

Pro informace o	Viz
Deaktivace posunu programu	*PDispOff - Deaktivuje posun programu na str 476*
Definice posunu programu pomocí známého rámce	*PDispSet - Aktivuje posun programu pomocí známého rámce na str 481*
Souřadné systémy	*Technická referenční příručka - Systémové parametry*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.167 PDispSet - Aktivuje posun programu pomocí známého rámce

Použití

`PDispSet` (*Program Displacement Set*) se používá k definování a aktivaci posunu programu pomocí známého rámce.

Posun programu se používá například když podobné vzorce pohybu jsou opakovány na několika různých místech v programu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `PDispSet`:

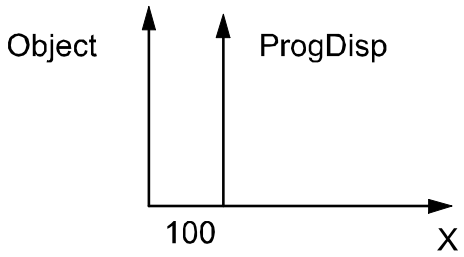
Příklad 1

```
VAR pose xp100 := [ [100, 0, 0], [1, 0, 0, 0] ];
...
PDispSet xp100;
```

Aktivace posunu programu `xp100` znamená, že:

- Souřadný systém `ProgDisp` je posunut o 100 mm od souřadného systému objektu ve směru kladné osy x (viz obrázek dole).
- Dokud je tento posun programu aktivní, všechny pozice budou posunuty o 100 mm ve směru osy x.

Obrázek ukazuje posun o 100 mm podél osy x.



xx0500002199

Argumenty

```
PDispSet DispFrame
```

`DispFrame`

Displacement Frame

Datový typ: `pose`

Posun programu je definován jako data typu `pose`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

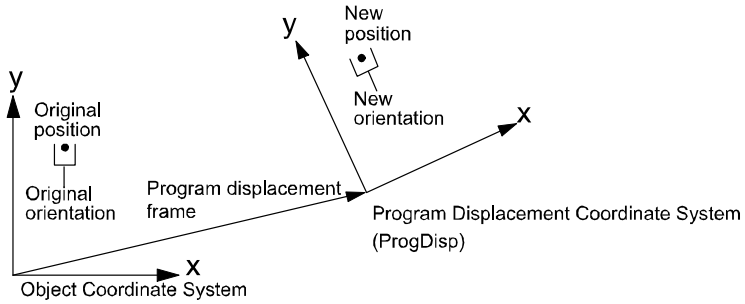
1.167 PDispSet - Aktivuje posun programu pomocí známého rámce

RobotWare - OS

Pokračování

Vykonávání programu

Posun programu zahrnuje překlad a/nebo rotaci souřadného systému `ProgDisp` ve vztahu k souřadnému systému objektu. Jelikož všechny pozice jsou vztaženy k souřadnému systému `ProgDisp`, všechny naprogramované pozice budou také posunuty. Viz obrázek dole, který ukazuje překlad a rotaci naprogramované pozice.



xx0500002204

Posun programu se aktivuje, když je vykonána instrukce `PDispSet` a zůstává aktivní až do doby aktivace jiného posunu programu (instrukce `PDispSet` nebo `PDispOn`) nebo do deaktivace posunu programu (instrukce `PDispOff`).

Pouze jeden posun program může být aktivní ve stejném čase. Posuny programu není možné k sobě přidávat pomocí `PDispSet`.

Posun programu se nastavuje automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Syntaxe

```
PDispSet
  [ DispFrame ':=' ] < expression (IN) of pose> ';'
```

Související informace

Pro informace o	Viz
Deaktivace posunu programu	*PDispOff - Deaktivuje posun programu na str 476*
Definice posunu programu pomocí dvou pozic	*PDispOn - Aktivuje posun programu na str 477*
Definice dat typu pose	*pose - Transformace souřadnic na str 1555*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Příklady použití posunu programu	*PDispOn - Aktivuje posun programu na str 477*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.168 ProcCall - Volá novou proceduru

Použití

Volání procedury se používá pro přenos vykonávání programu k jiné proceduře. Když byla procedura plně vykonána, vykonávání programu pokračuje s instrukcí následující po volání procedury.

Obvykle je možné odeslat řadu argumentů k nové proceduře. Kontrolují chování procedury a umožňují, aby stejná procedura mohla být použita pro různé účely.

Základní příklady

Následující příklady názorně ukazují instrukci `ProcCall`:

Příklad 1

```
weldpipe1;
```

Volá proceduru `weldpipe1` .

Příklad 2

```
errormessage;
Set do1;
...
PROC errormessage()
  TPWrite "ERROR";
ENDPROC
```

Procedura `errormessage` je volána. Když je tato procedura připravena, vykonávání programu se vrací k instrukci následující po volání procedury, `Set do1`.

Argumenty

```
Procedure { Argument }
```

`Procedure`

Identifier

Jméno procedury, která bude volána.

`Argument`

Datový typ: V souladu s deklarací procedury.

Argumenty procedury (v souladu s parametry procedury).

Základní příklady

Základní příklady instrukce `ProcCall` jsou názorně uvedeny dole.

Příklad 1

```
weldpipe2 10, lowspeed;
```

Volá proceduru `weldpipe2` včetně dvou argumentů.

Příklad 2

```
weldpipe3 10 \speed:=20;
```

Volá proceduru `weldpipe3` včetně jednoho povinného a jednoho volitelného argumentu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.168 ProcCall - Volá novou proceduru
RobotWare - OS
Pokračování

Omezení

Argumenty procedury musí souhlasit s jejími parametry:

- Všechny povinné argumenty musí být zahrnuty.
- Musí být umístěny ve stejném pořadí.
- Musí být stejného datového typu.
- Musí být správného typu s ohledem na přístupový režim (vstup, proměnná nebo perzistent).

Rutina může volat rutinu, která obratem volá další rutinu. Rutina může také volat samu sebe, to je rekurzivní volání. Řada povolených úrovní rutiny závisí na počtu parametrů. Obvykle je povoleno více než 10 úrovní.

Syntaxe

```
<procedure> [ <argument list> ] ';'
```

Související informace

Pro informace o	Viz
Argumenty, parametry	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.169 ProcerrRecovery - Generovat a zotavit z chyby procesního pohybu

Použití

`ProcerrRecovery` se může používat pro generování procesní chyby během pohybu robotu a získat tak možnost k ošetření chyby a restartování procesu a pohybu od ERROR handleru.

Základní příklady

Následující příklady názorně ukazují instrukci `ProcerrRecovery`:

Viz také *Další příklady na str 486*.

Příklady dole nejsou realistické, ale jsou ukázány z pedagogických důvodů.

Příklad 1

```
MoveL p1, v50, z30, tool2;
ProcerrRecovery \SyncOrgMoveInst;
MoveL p2, v50, z30, tool2;
ERROR
  IF ERRNO = ERR_PATH_STOP THEN
    StartMove;
    RETRY;
  ENDIF
```

Pohyb robotu se zastaví na své cestě k `p1` a vykonávání programu se přenese k `ERROR` handleru v rutině, která vytvořila aktuální dráhu, na které chyba vznikla, v tomto případě dráhu k `MoveL p1`. Pohyb je restartován s `StartMove` a vykonávání pokračuje s `RETRY`.

Příklad 2

```
MoveL p1, v50, fine, tool2;
ProcerrRecovery \SyncLastMoveInst;
MoveL p2, v50, z30, tool2;
ERROR
  IF ERRNO = ERR_PATH_STOP THEN
    StartMove;
    RETRY;
  ENDIF
```

Pohyb robotu se okamžitě zastaví na své cestě k `p2`. Vykonávání programu se přenese k `ERROR` handleru v rutině, kde se program právě vykonává nebo se měla vykonávat pohybová instrukce, když se chyba objevila, v tomto případě `MoveL p2`. Pohyb je restartován s `StartMove` a vykonávání pokračuje s `RETRY`.

Argumenty

```
ProcerrRecovery[\SyncOrgMoveInst] | [\SyncLastMoveInst]
     [\ProcSignal]
```

`[\SyncOrgMoveInst]`

Datový typ: `switch`

Chyba může být ošetřena v rutině, která vytvořila aktuální dráhu, na které chyba vznikla.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.169 ProcerrRecovery - Generovat a zotavit z chyby procesního pohybu

RobotWare - OS

Pokračování

[\SyncLastMoveInst]

Datový typ: switch

Chyba může být ošetřena v rutině, kde program momentálně vykonává pohybovou instrukci, když chyba vznikla.

Jestliže program momentálně nevykonává pohybovou instrukci, když chyba vznikla, potom bude přenos vykonávání k ERROR handleru opožděn, dokud program nevykoná další pohybovou instrukci. To znamená, že přenos k ERROR handleru bude opožděn, jestliže robot je ve stop bodu nebo mezi prefetch bodem a středem rohové dráhy. Chyba může být ošetřena v této rutině.

[\ProcSignal]

Datový typ: signaldo

Volitelný parametr, který umožňuje uživateli zapnout/vypnout použití instrukce. Jestliže se tento parametr používá a hodnota signálu je 0, řešitelná chyba bude vyhozena a žádná procesní chyba nebude generována.

Vykonávání programu

Vykonání ProcerrRecovery v nepřetržitém režimu má následující výsledek:

- Robot je náhle zastaven na své dráze.
- Proměnná ERRNO je nastavena na ERR_PATH_STOP.
- Vykonávání je přeneseno k některému ERROR handleru podle pravidel pro asynchronně pozvednuté chyby.

Tato instrukce nedělá nic v žádném krokovém režimu.

Popis asynchronně pozvednutých chyb, které jsou generovány s ProcerrRecovery najdete v RAPID kernel reference/Zotavení z chyb/Asynchronně pozvednuté chyby.

ProcerrRecovery se může také používat v systému MultiMove k přenášení vykonávání k ERROR handleru v několika programových úlohách při běhu v synchronizovaném režimu.

Další příklady

Více příkladů jak používat instrukci ProcerrRecovery je názorně uvedeno dole.

Příklad s ProcerrRecovery\SyncOrgMoveInst

```
MODULE user_module
  VAR intnum proc_sup_int;

  PROC main()
    ...
    MoveL p1, v1000, fine, tool1;
    do_process;
    ...
  ENDPROC
  PROC do_process()
    my_proc_on;
    MoveL p2, v200, z10, tool1;
    MoveL p3, v200, fine, tool1;
    my_proc_off;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
    ERROR
      IF ERRNO = ERR_PATH_STOP THEN
        my_proc_on;
        StartMove;
        RETRY;
      ENDIF
    ENDPROC

    TRAP iprocfail
      my_proc_off;
      ProcerrRecovery \SyncOrgMoveInst;
    ENDTRAP

    PROC my_proc_on()
      SetDO do_myproc, 1;
      CONNECT proc_sup_int WITH iprocfail;
      ISignalDI di_proc_sup, 1, proc_sup_int;
    ENDPROC

    PROC my_proc_off()
      SetDO do_myproc, 0;
      IDelete proc_sup_int;
    ENDPROC
  ENDMODULE
```

Asynchronně pozvednuté chyby generované `ProcerrRecovery` **s přepínačem** `\SyncOrgMoveInst` **mohou být v tomto příkladu ošetřeny v do_process rutiny, protože dráha, na které se chyba objevila, je vždy vytvořena v rutině d**`o_process`**.**

Procesní tok se spouští nastavením signálu `do_myproc` **na** `1`**. Signál** `di_proc_sup` **dohlíží na proces a asynchronní chyba je pozvednuta, jestliže** `di_proc_sup` **je** `1`**. V tomto jednoduchém příkladu je chyba vyřešena nastavením** `do_myproc` **znovu na** `1` **před obnovením pohybu.**

Příklad s ProcerrRecovery\SyncLastMoveInst

```
MODULE user_module
  PROC main()
    ...
    MoveL p1, v1000, fine, tool1;
    do_process;
    ...
  ENDPROC
  PROC do_process()
    proc_on;
    proc_move p2, v200, z10, tool1;
    proc_move p3, v200, fine, tool1;
    proc_off;
  ERROR
    IF ERRNO = ERR_PATH_STOP THEN
      StorePath;
      p4 := CRobT(\Tool:=tool1);
      ! Move to service station and fix the problem
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.169 ProcerrRecovery - Generovat a zotavit z chyby procesního pohybu
RobotWare - OS
Pokračování

```
      MoveL p4, v200, fine, tool1;
      RestoPath;
      proc_on;
      StartMoveRetry;
    ENDIF
  ENDPROC
ENDMODULE

MODULE proc_module (SYSMODULE, NOSTEPIN)
  VAR intnum proc_sup_int;
  VAR num try_no := 0;

TRAP iprocfail
    proc_off;
    ProcerrRecovery \SyncLastMoveInst;
  ENDTRAP

  PROC proc_on()
    SetDO do_proc, 1;
    CONNECT proc_sup_int WITH iprocfail;
    ISignalDI di_proc_sup, 1, proc_sup_int;
  ENDPROC

  PROC proc_off()
    SetDO do_proc, 0;
    IDelete proc_sup_int;
  ENDPROC

PROC proc_move (robtarget ToPoint, speeddata Speed, zonedata Zone,
     PERS tooldata Tool)
    MoveL ToPoint, Speed, Zone, Tool;
    ERROR
      IF ERRNO = ERR_PATH_STOP THEN
        try_no := try_no + 1;
        IF try_no < 4 THEN
          proc_on;
          StartMoveRetry;
        ELSE
          RaiseToUser \Continue;
      ENDIF
    ENDPROC
ENDMODULE
```

Asynchronně pozvednuté chyby generované od `ProcerrRecovery` **s přepínačem** `\SyncLastMoveInst` **mohou být v tomto příkladu ošetřeny v rutině** `proc_move`**, protože všechny pohybové instrukce jsou vždy vytvořeny v rutině proc_move. Když ukazatel programu je v rutině** `do_process`**, přenos k** `ERROR` **handleru bude opožděn až do proběhnutí dalšího** `MoveL` **v rutině** `proc_move`**. Všimněte si, že pohyby jsou vždy zastaveny okamžitě.**

Procesní tok se spouští nastavením signálu `do_myproc` **na** `1`**. Signál** `di_proc_sup` **dohlíží na proces a asynchronní chyba je pozvednuta, jestliže** `di_proc_sup` **je** `1`**.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

V tomto jednoduchém příkladu je chyba vyřešena nastavením `do_myproc` znovu na `1` před obnovením pohybu.

Při používání předdefinované rutiny `NOSTEPIN` doporučujeme použít parametr volitelného spínače `\SyncLastMoveInst`, protože potom může předdefinovaná rutina udělat rozhodnutí ošetřit některou chybovou situaci v rutině, zatímco ostatní musí být ošetřeny koncovým uživatelem.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_PROCSIGNAL_OFF`, jestliže se používá volitelný parametr `\ProcSignal` a jestliže signál je vypnut, když je instrukce vykonávána.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Omezení

Obnovení po asynchronně pozvednutých procesních chybách může být provedeno pouze v případě, že pohybová úloha s procesní pohybovou instrukcí je vykonávána na úrovni základny, když se procesní chyba objeví. Takže obnova po chybě se nemůže provést, jestliže programová úloha s procesní instrukcí probíhá v:

- jakékoliv událostní rutině
- jakémkoliv handleru rutiny (`ERROR`, `BACKWARD` nebo `UNDO`)
- na uživatelské úrovni vykonávání (servisní rutina)

Viz *RAPID referenční příručka - RAPID kernel*, *Obnova po chybách*, *Asynchronně pozvednuté chyby*.

Jestliže se nepoužívá žádný chybový handler s `StartMove` + `RETRY` nebo `StartMoveRetry`, vykonávání programu se zastaví. Reset je možný jedině cestou provedení PP na main.

Syntaxe

```
ProcerrRecovery
  [ '\' SyncOrgMoveInst ] | [' \' SyncLastMoveInst ]
  [ '\' ProcSignal' :=' ] < variable (VAR) of signaldo > ';'
```

Související informace

Pro informace o	Viz
Obslužné programy pro řešení chyb	*Technická referenční příručka - Přehled RAPID*
Asynchronně pozvednuté chyby	*RAPID referenční příručka - RAPID kernel - Obnova po chybách*
Rozšiřuje chybu na uživatelskou úroveň	*RaiseToUser - Rozšiřuje chybu na uživatelskou úroveň na str 515*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.169 ProcerrRecovery - Generovat a zotavit z chyby procesního pohybu

RobotWare - OS

Pokračování

Pro informace o	Viz
Obnovit pohyb a vykonávání programu	*StartMoveRetry - Restartuje pohyb robotu a vykonávání na str 710*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.170 PrxActivAndStoreRecord - Aktivovat a uložit zaznamenaná profilová data

Použití

`PrxActivAndStoreRecord` se používá k aktivaci zaznamenaných profilových dat a jejich uložení do souboru.

Může se použít namísto volání `PrxActivRecord` a `PrxStoreRecord`.

Základní příklad

```
PrxActivAndStoreRecord SSYNC1, 1, "profile.log";
```

Profil pohybu senzoru aktivován a uložen do souboru *profile.log*.

Argumenty

```
PrxActivAndStoreRecord MechUnit Delay File_name
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

`Delay`

Datový typ: `num`

Prodleva v sekundách se může používat k posunutí záznamu v čase. Musí to být mezi 0,01 a 0,1. Při dané hodnotě 0 nebude přidána žádná prodleva. Prodleva se neukládá do profilu, je pouze použita pro aktivaci. Jestliže prodleva by měla být použita společně s uloženým profilem, prodleva musí být stanovena v instrukci `PrxUseFileRecord`.

`File_name`

Datový typ: `string`

Jméno souboru, kde je uložen profil.

Vykonávání programu

`PrxActivAndStoreRecord` musí být vykonáno alespoň 0,2 sekundy před spuštěním pohybu senzoru, jestliže záznam má být použit pro synchronizaci.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v chybovém handleru. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_ACTIV_PROF`	Chyba v aktivovaném profilu
`ERR_STORE_PROF`	Chyba v uloženém profilu
`ERR_USE_PROF`	Chyba v použitém profilu

Syntaxe

```
PrxActivAndStoreRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Delay ':=' ] < expression (IN) of num > ','
  [ File_name ':=' ] < expression (IN) of string > ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.170 PrxActivAndStoreRecord - Aktivovat a uložit zaznamenaná profilová data

Pokračování

Související informace

Pro informace o	Viz
Aktivovat zaznamenaná profilová data	*PrxActivRecord - Aktivovat zaznamenaná profilová data na str 493*
Deaktivovat záznam	*PrxDeactRecord - Deaktivovat záznam na str 496*
Uložit zaznamenaná profilová data	*PrxStoreRecord - Uložit zaznamenaná profilová data na str 506*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.171 PrxActivRecord - Aktivovat zaznamenaná profilová data

Použití

`PrxActivRecord` se používá pro aktivaci záznamu, který byl právě pořízen, aby mohl být použit bez předchozího ukládání.

Základní příklad

```
PrxActivRecord SSYNC1, 0;
WaitTime 0.2;
SetDO do_startstop_machine, 1;
!Work synchronized with sensor
...
SetDO do_startstop_machine, 0;
```

Záznam senzoru je aktivován a použit pro předpověď pohybu senzoru, jakmile je záznam připraven.

Argumenty

```
PrxActivRecord MechUnit Delay
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

`Delay`

Datový typ: `num`

Prodleva v sekundách se může používat k posunu záznamu v čase. Musí to být mezi 0,01 a 0,1. Při dané hodnotě 0 nebude přidána žádná prodleva.

Vykonávání programu

`PrxActivRecord` musí být vykonán alespoň 0,2 sekundy před spuštěním pohybu dopravníku.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v chybovém handleru. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_ACTIV_PROF`	Chyba v aktivovaném profilu
`ERR_STORE_PROF`	Chyba v uloženém profilu
`ERR_USE_PROF`	Chyba v použitém profilu

Syntaxe

```
PrxActivRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Delay ':=' ] < expression (IN) of num > ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.171 PrxActivRecord - Aktivovat zaznamenaná profilová data

Pokračování

Související informace

Pro informace o	Viz
Aktivovat a uložit zaznamenaná profilová data	*PrxActivAndStoreRecord - Aktivovat a uložit zaznamenaná profilová data na str 491*
Deaktivovat záznam	*PrxDeactRecord - Deaktivovat záznam na str 496*
Uložit zaznamenaná profilová data	*PrxStoreRecord - Uložit zaznamenaná profilová data na str 506*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.172 PrxDbgStoreRecord - Uložit a ladit zaznamenaná profilová data

Použití

`PrxDbgStoreRecord` se používá k uložení neaktivovaného záznamu pro ladění.

Může se používat k porovnání záznamů a kontrole opakovatelnosti.

Základní příklad

```
PrxDbgStoreRecord SSYNC1, "debug_profile.log";
```

Ukládá záznam do souboru *debug_profile.log*.

Argumenty

```
PrxDbgStoreRecord MechUnit Filename
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

`File_name`

Datový typ: `string`

Jméno souboru, kde je uložen záznam.

Syntaxe

```
PrxDbgStoreRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ File_name ':=' ] < expression (IN) of string > ';'
```

Související informace

Pro informace o	Viz
Aktivovat a uložit zaznamenaná profilová data	*PrxActivAndStoreRecord - Aktivovat a uložit zaznamenaná profilová data na str 491*
Aktivovat zaznamenaná profilová data	*PrxActivRecord - Aktivovat zaznamenaná profilová data na str 493*
Uložit zaznamenaná profilová data	*PrxStoreRecord - Uložit zaznamenaná profilová data na str 506*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.173 PrxDeactRecord - Deaktivovat záznam

Použití

`PrxDeactRecord` se používá pro deaktivaci záznamu.

Základní příklad

```
PrxDeactRecord SSYNC1;
```

Záznam pohybu senzoru je deaktivován a již dále nepoužíván pro předpověď pohybu senzoru. Záznam může být znovu aktivován.

Argumenty

```
PrxDeactRecord MechUnit
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

Omezení

`PrxDeactRecord` by neměl být volán během synchronizace.

Syntaxe

```
PrxDeactRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ';'
```

Související informace

Pro informace o	Viz
Aktivovat zaznamenaná profilová data	*PrxActivRecord - Aktivovat zaznamenaná profilová data na str 493*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.174 PrxResetPos - Resetovat nulovou pozici senzoru

Použití

`PrxResetPos` se používá pro reset nulové pozice senzoru.

Pozice senzoru je resetována kvůli funkčnosti synchronizace a zaznamenanému souboru, ale hodnota I/O signálu není resetována. Tato instrukce se používá pro softwarový reset vstupu senzoru, kde není k dispozici žádný synch přepínač pro reset I/O signálu.

Základní příklad

```
PrxResetPos SSYNC1;
```

Pozice senzoru je nastavena na nulu.

Argumenty

```
PrxResetPos MechUnit
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

Vykonávání programu

Jednotka senzoru musí být zastavena (v požadované nulové pozici) před voláním `PrxResetPos`.

Omezení

Nebude se používat s deskou DSQC 377A.

Tato instrukce je rovnocenná se synch přepínačem. Okno ručního posuvu (jogging) by mělo ukazovat 0.0 jako dodatečnou pozici osy po této instrukci.

Syntaxe

```
PrxResetPos
  [ MechUnit ':=' ] < expression (IN) of mechunit> ';'
```

Související informace

Pro informace o	Viz
Nastavit referenční pozici pro senzor	*PrxSetPosOffset - Nastavit referenční pozici pro senzor na str 499*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.175 PrxResetRecords - Resetovat a deaktivovat všechny záznamy

Použití

`PrxResetRecords` se používá pro reset a deaktivaci všech záznamů.

Základní příklad

```
PrxResetRecords SSYNC1;
```

Záznam pohybu senzoru je deaktivován a již dále nepoužíván pro předpověď pohybu senzoru a zaznamenaná data jsou odstraněna.

Argumenty

```
PrxResetRecords MechUnit
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

Vykonávání programu

`PrxResetRecords` musí být vykonán alespoň 0,2 sekundy před spuštěním pohybu dopravníku.

Syntaxe

```
PrxResetRecords
  [ MechUnit ':=' ] < expression (IN) of mechunit> ';'
```

Související informace

Pro informace o	Viz
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.176 PrxSetPosOffset - Nastavit referenční pozici pro senzor

Použití

`PrxSetPosOffset` se používá pro nastavení referenční pozice pro senzor.

Pozice senzoru se nastavuje pro referenci funkčnosti synchronizace a zaznamenaného souboru. Tato funkce se používá pro softwarové nastavení reference senzoru, kde není dostupný žádný synch přepínač pro reset I/O signálu.

Základní příklad

```
PrxSetPosOffset SSYNC1, reference;
```

Pozice senzoru je nastavena na referenční hodnotu.

Argumenty

```
PrxSetPosOffset MechUnit Reference
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

`Reference`

Datový typ: `num`

Reference v metrech (nebo jednotce senzoru). Musí být mezi -5000 a 5000.

Vykonávání programu

Jednotka senzoru musí být zastavena před voláním `PrxSetPosOffset`.

Omezení

Nebude se používat s deskou DSQC 377A.

Syntaxe

```
PrxSetPosOffset
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Reference ':=' ] < expression (IN) of num > ';'
```

Související informace

Pro informace o	Viz
Resetovat nulovou pozici senzoru	*PrxResetPos - Resetovat nulovou pozici senzoru na str 497*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.177 PrxSetRecordSampleTime - Nastavit vzorkový čas pro záznam profilu

Použití

`PrxSetRecordSampleTime` se používá pro nastavení vzorkového času, v sekundách, pro záznam profilu.

Výchozí vzorkový čas je brán ze systémového parametru *Pos Update time*, patřícího k typu *CAN interface* v tématu *Process*. Všimněte si, že *Pos Update time* určuje vzorkový čas v milisekundách, zatímco `PrxSetRecordSampleTime` určuje vzorkový čas v sekundách.

Max počet vzorků v zaznamenaném profilu je 300. Jestliže je záznam delší než 3000 * *Pos Update time*, čas vzorku musí být prodloužen.

Základní příklad

Bude pořízen záznam v délce 12 sekund. Čas vzorku nemůže být méně než 12/300 = 0,04. Čas vzorku je tedy nastaven na 0,04 sekundy.

```
PrxSetRecordSampleTime SSYNC1, 0.04;
```

Argumenty

```
PrxSetRecordSampleTime MechUnit SampleTime
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

`SampleTime`

Datový typ: `num`

Čas vzorku v sekundách. Čas vzorku musí být mezi 0,01 a 0,1.

Syntaxe

```
PrxSetRecordSampleTime
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ SampleTime ':=' ] < expression (IN) of num> ';'
```

Související informace

Pro informace o	Viz
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.178 PrxSetSyncalarm - Nastavit chování synch alarmu

Použití

`PrxSetSyncalarm` se používá k nastavení chování *sync_alarm_signal* na impuls během určeného času.

Jestliže je spuštěn synch alarm, *Sync_alarm_signal* pulsuje během doby určené instrukcí `PrxSetSyncalarm`. Může být také nastaveno na žádný impuls, tj. signál pokračuje jako vysoký.

Výchozí délka impulsu je 1 sekunda.

Základní příklady

Příklad 1

```
PrxSetSyncalarm SSYNC1 \time:=2;
```

Nastavuje délku impulsu na *sync_alarm_signal* na 2 sekundy.

Příklad 2

```
PrxSetSyncalarm SSYNC1 \NoPulse;
```

Jestliže je synch alarm spuštěn, *sync_alarm_signal* je nastaven (bez impulsu).

Argumenty

```
PrxSetSyncalarm MechUnit [\Time] | [\NoPulse]
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

`[\Time]`

Datový typ: `num`

Délka impulsu v sekundách. Musí být mezi 0,1 a 60.

Jestliže `\Time` je nastaven na více než 60, nepoužívá se žádný impuls (stejný efekt jako při používání `\NoPulse`).

`[\NoPulse]`

Datový typ: `switch`

Nepoužívá se žádný impuls. Signál je nastaven do vykonávání instrukce `SupSyncSensorOff`:

Syntaxe

```
PrxSetSyncalarm
  [ MechUnit ':=' ] < expression (IN) of mechunit>
  [ '\' Time ':=' < expression (IN) of num > ]
  | [ '\' NoPulse ] ';'
```

Související informace

Pro informace o	Viz
SupSyncSensorOff - Zastavit dohled synchronizovaného senzoru	*SupSyncSensorOff - Zastavit dohled synchronizovaného senzoru na str 749*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.178 PrxSetSyncalarm - Nastavit chování synch alarmu

Pokračování

Pro informace o	Viz
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.179 PrxStartRecord - Zaznamenat nový profil

Použití

PrxStartRecord se používá pro reset všech profilových dat a záznam nového profilu pohybu senzoru, jakmile je nastaven *sensor_start_signal*.

Aby bylo možné udělat záznam, je důležité nejprve udělat spojení k senzoru (mechanická jednotka, jejíž rychlost ovlivňuje rychlost robotu). To znamená, že instrukce `WaitSensor` musí být vykonána před začátkem záznamu.

Základní příklad

```
ActUnit SSYNC1;
WaitSensor SSYNC1;
PrxStartRecord SSYNC1, 1, PRX_PROFILE_T1;
WaitTime 0.2;
SetDO do_startstop_machine 1;
```

Signál *do_startstop_machine* v tomto příkladu spouští pohyb senzoru. Profil senzoru je zaznamenán, jakmile stroj nastaví signál *sensor_start_signal*.

Argumenty

```
PrxStartRecord MechUnit, Record_duration, Profile_type
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

`Record_duration`

Datový typ: `num`

Určuje dobu trvání záznamu v sekundách. Musí to být mezi 0,1 a *Pos Update time* * 300. Jestliže je použita hodnota 0, instrukci `PrxStopRecord` je nutné použít pro zastavení záznamu.

`Profile_type`

Datový typ: `num`

Možné hodnoty a jejich vysvětlivky jsou uvedeny dole:

Hodnota	Popis
`PRX_INDEX_PROF`	Záznam spustil *sensor_start_signal*.
`PRX_START_ST_PR`	Pohyb spuštění a zastavení může být zaznamenán. *sensor_start_signal* se používá pro záznam pohybu spuštění a *sensor_stop_signal* se používá pro záznam pohybu zastavení.
`PRX_STOP_ST_PROF`	Stejné jako u `PRX_START_ST_PR`, pouze odlišné pořadí signálů. *sensor_stop_signal* je použit jako první.
`PRX_STOP_M_PROF`	Záznam spustil *sensor_stop_signal*.
`PRX_HPRESS_PROF`	Pro záznam hydraulického lisu (kde nulová pozice senzoru odpovídá otevření lisu).
`PRX_PROFILE_T1`	Pro záznam IMM nebo jiného stroje (kde nulová pozice senzoru odpovídá zavření lisu).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.179 PrxStartRecord - Zaznamenat nový profil

Pokračování

Vykonávání programu

`PrxStartRecord` musí být vykonán alespoň 0,2 sekundy před spuštěním pohybu senzoru.

Syntaxe

```
PrxStartRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Record_duration ':=' ] < expression (IN) of num > ','
  [ Profile_type ':=' ] < expression (IN) of num > ';'
```

Související informace

Pro informace o	Viz
Zastavit záznam profilu	*PrxStopRecord - Zastavit záznam profilu na str 505*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.180 PrxStopRecord - Zastavit záznam profilu

Použití

`PrxStopRecord` se používá pro zastavení záznamu profilu.

Mělo by se používat vždy, když `PrxStartRecord` má `Record_duration` nastaveno na 0.

Základní příklad

```
ActUnit SSYNC1;
WaitSensor SSYNC1;
PrxStartRecord SSYNC1, 0, PRX_PROFILE_T1;
WaitTime 0.2;
SetDo do_startstop_machine 1;
WaitTime 2;
PrxStopRecord SSYNC1;
```

Signál *do_startstop_machine* v tomto příkladu spouští pohyb senzoru. Profil pohybu senzoru je zaznamenán, jakmile *sensor_start_signal* je nastaven a po dvou sekundách je záznam zastaven s instrukcí `PrxStopRecord`.

Argumenty

```
PrxStopRecord MechUnit
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

Syntaxe

```
PrxStopRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ';'
```

Související informace

Pro informace o	Viz
Zaznamenat nový profil	*PrxStartRecord - Zaznamenat nový profil na str 503*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.181 PrxStoreRecord - Uložit zaznamenaná profilová data

Použití

`PrxStoreRecord` se používá k uložení aktivovaného záznamu do souboru.

Základní příklad

```
ActUnit SSYNC1;
WaitSensor SSYNC1;
PrxStartRecord SSYNC1, 0, PRX_PROFILE_T1;
WaitTime 0.2;
SetDo do_startstop_machine 1;
WaitTime 2;
PrxStopRecord SSYNC1;
PrxActivRecord SSYNC1;
SetDo do_startstop_machine 0;
PrxStoreRecord SSYNC1, 0, "profile.log";
```

Profil pohybu senzoru je zaznamenán, jakmile *sensor_start_signal* je nastaven a je uložen do souboru *profile.log*.

Argumenty

```
PrxStoreRecord MechUnit Delay Filename
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

`Delay`

Datový typ: `num`

Prodleva v sekundách se může používat k posunutí záznamu v čase. Musí to být mezi 0,01 a 0,1. Při dané hodnotě 0 nebude přidána žádná prodleva. Prodleva se neukládá do profilu, je pouze použita pro aktivaci. Jestliže prodleva by měla být použita společně s uloženým profilem, prodleva musí být stanovena v instrukci `PrxUseFileRecord`.

`File_name`

Datový typ: `string`

Jméno souboru, kde je uložen profil.

Omezení

Záznam musí být aktivován před voláním `PrxStoreRecord`.

Syntaxe

```
PrxStoreRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Delay ':=' ] < expression (IN) of num > ','
  [ File_name ':=' ] < expression (IN) of string > ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Aktivovat zaznamenaná profilová data	*PrxActivRecord - Aktivovat zaznamenaná profilová data na str 493*
Deaktivovat záznam	*PrxDeactRecord - Deaktivovat záznam na str 496*
Použít zaznamenaná profilová data	*PrxUseFileRecord - Použít zaznamenaná profilová data na str 508*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.182 PrxUseFileRecord - Použít zaznamenaná profilová data

Použití

`PrxUseFileRecord` se používá pro načtení a aktivaci záznamu ze souboru pro synchronizaci senzoru.

Základní příklad

```
PrxUseFileRecord SSYNC1, 0, "profile.log";
WaitTime 0.2;
SetDo do_startstop_machine 1;
!Work synchronized with sensorWork synchronized with sensor
...
SetDo do_startstop_machine 0;
```

Argumenty

```
PrxUseFileRecord MechUnit Delay Filename
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

`Delay`

Datový typ: `num`

Prodleva v sekundách se může používat k posunu záznamu v čase. Musí to být mezi 0,01 a 0,1. Při dané hodnotě 0 nebude přidána žádná prodleva.

`File_name`

Datový typ: `string`

Jméno souboru, kde je uložen profil.

Vykonávání programu

`PrxUseFileRecord` musí být vykonán alespoň 0,2 sekundy před spuštěním pohybu dopravníku.

Syntaxe

```
PrxUseFileRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Delay ':=' ] < expression (IN) of num > ','
  [ File_name ':=' ] < expression (IN) of string > ';'
```

Související informace

Pro informace o	Viz
Aktivovat zaznamenaná profilová data	*PrxActivRecord - Aktivovat zaznamenaná profilová data na str 493*
Deaktivovat záznam	*PrxDeactRecord - Deaktivovat záznam na str 496*
Uložit zaznamenaná profilová data	*PrxStoreRecord - Uložit zaznamenaná profilová data na str 506*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.183 PulseDO - Generuje impuls na digitálním výstupním signálu

Použití

`PulseDO` is se používá pro generování impulsu na digitálním výstupním signálu.

Základní příklady

Následující příklady názorně ukazují instrukci `PulseDO`:

Příklad 1

```
PulseDO do15;
```

Impuls s délkou 0,2 sekundy je generován na výstupním signálu `do15`.

Příklad 2

```
PulseDO \PLength:=1.0, ignition;
```

Impuls s délkou 1,0 sekundy je generován na signálu `ignition`.

Příklad 3

```
! Program task MAIN
PulseDO \High, do3;
! At almost the same time in program task BCK1
PulseDO \High, do3;
```

Kladný impuls (hodnota 1) je generován na signálu `do3` od dvou programových úloh současně. Bude to mít za následek jeden kladný impuls o délce větší než výchozí 0,2 sekundy nebo dva kladné impulsy, jeden po druhém, o délce impulsu 0,2 sekundy.

Argumenty

```
PulseDO [ \High ] [ \PLength ] Signal
```

`[ \High ]`

High level

Datový typ: `switch`

Určuje, že hodnota signálu by měla být vždy nastavena na vysoko (hodnota 1), když je instrukce vykonávána nezávisle na svém aktuálním stavu.

`[ \PLength ]`

Pulse Length

Datový typ: `num`

Délka impulsu v sekundách (0,001 - 2000 s). Jestliže je argument vypuštěn, bude generován impuls 0,2 s.

`Signal`

Datový typ: `signaldo`

Jméno signálu, na kterém bude generován impuls.

Pokračování na další straně

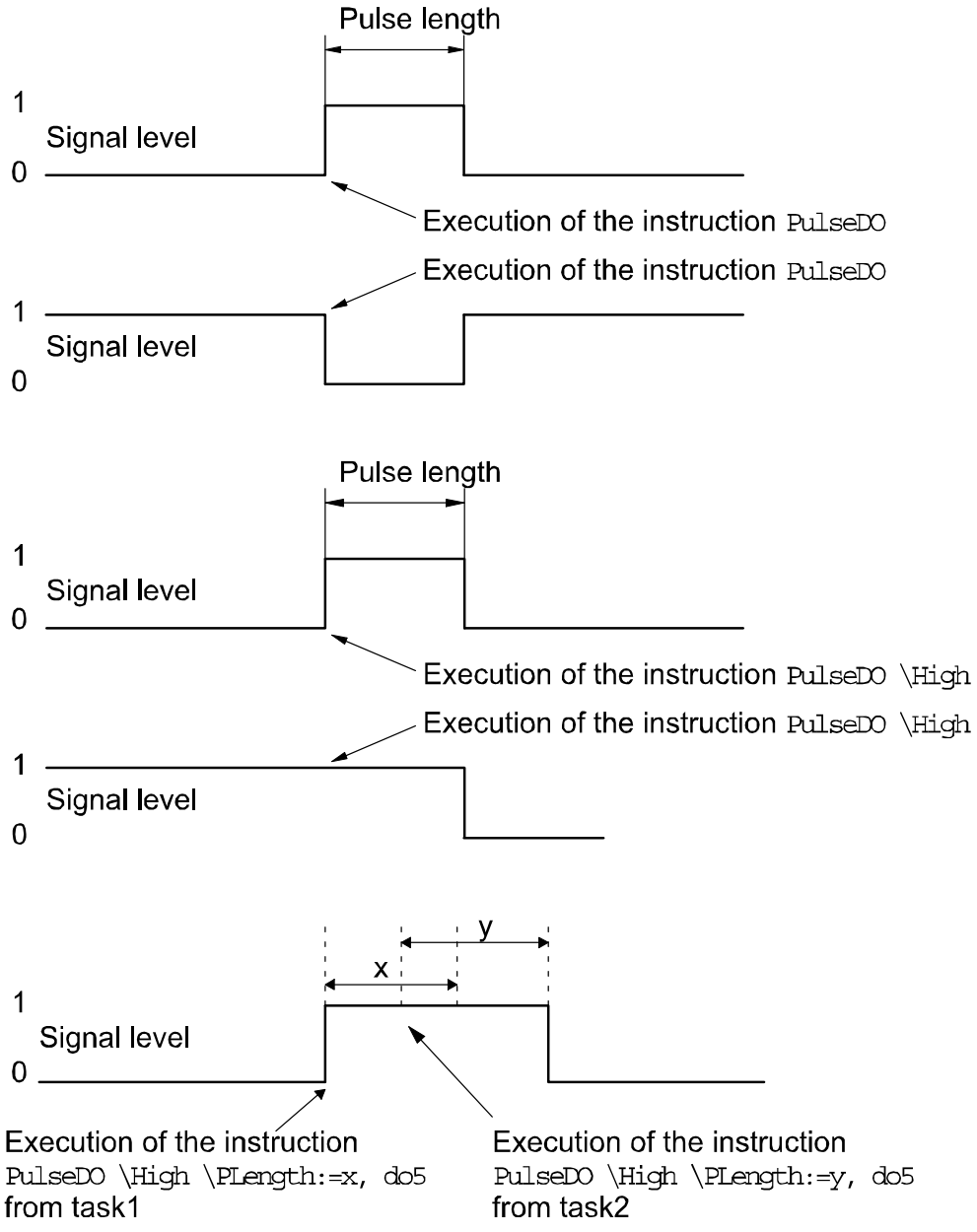
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.183 PulseDO - Generuje impuls na digitálním výstupním signálu
RobotWare - OS
Pokračování

Vykonávání programu

Další instrukce po `PulseDO` je vykonána přímo po spuštění impulsu. Impuls může být potom nastaven/resetován bez ovlivnění zbytku vykonávání programu.

Obrázek dole ukazuje příklady generování impulsů na digitálním výstupním signálu.



xx0500002217

Další instrukce po je vykonána přímo po spuštění impulsu. Impuls může být potom nastaven/resetován bez ovlivnění zbytku vykonávání programu.

Omezení

Délka impulsu má vypnuté rozlišení 0,001 s. Naprogramované hodnoty, které se od tohoto liší, jsou zaokrouhleny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_NO_ALIASIO_DEF jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.

ERR_NORUNUNIT jestliže není žádný kontakt s jednotkou I/O.

ERR_SIG_NOT_VALID jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
PulseDO
  [ '\' High]
  [ '\' PLength ':=' < expression (IN) of num >] ','
  [ Signal ':=' ] < variable (VAR) of signaldo > ';'
```

Související informace

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.184 RAISE - Volá chybový handler

Použití

RAISE se používá pro vytvoření chyby v programu a následného volání chybového handleru rutiny.

RAISE může být také použít v chybovém handleru k rozšíření aktuální chyby k chybovému handleru volající rutiny.

Instrukci je možné použít například ke skoku zpět na vyšší úroveň ve struktuře programu, např. k chybovému handleru v main rutině, jestliže chyba se objevuje na nižší úrovni.

Základní příklady

Následující příklad názorně ukazuje instrukci RAISE:

Viz také *Další příklady na str 513*.

Příklad 1

```
MODULE MainModule .
VAR errnum ERR_MY_ERR := -1;

PROC main()
  BookErrNo ERR_MY_ERR;

  IF di1 = 0 THEN
    RAISE ERR_MY_ERR;
  ENDIF

ERROR
  IF ERRNO = ERR_MY_ERR THEN
    TPWrite "di1 equals 0";
  ENDIF

ENDPROC

ENDMODULE
```

U této implementace di1 je rovno 0 a je považováno za chybu. RAISE si vynutí vykonávání v chybovému handleru. Na tomto příkladu uživatel vytvořil své vlastní chybové číslo pro ošetření konkrétní chyby.

Argumenty

```
RAISE [ Error no. ]
```

Error no.

Datový typ: errnum

Chybové číslo: Jakékoliv číslo mezi 1 a 90, které může chybový handler použít k nalezení chyby, která se objevila (systémová proměnná ERRNO).

Je také možné rezervovat chybové číslo mimo rozsah 1-90 s instrukcí BookErrNo.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Chybové číslo musí být určeno mimo chybový handler v instrukci RAISE, aby vykonávání bylo přeneseno k chybovému handleru této rutiny.

Jestliže instrukce je přítomna v chybovém handleru rutiny, potom je chyba rozšířena k chybovému handleru volající rutiny. V tomto případě nemusí být chybové číslo určováno.

Další příklady

Více příkladů instrukce RAISE je názorně uvedeno dole.

Příklad 1

```
MODULE MainModule
VAR num value1 := 10;
VAR num value2 := 0;

PROC main()
  routine1;

ERROR
  IF ERRNO = ERR_DIVZERO THEN
    value2 := 1;
    RETRY;
  ENDIF
ENDPROC

PROC routine1()
  value1 := 5/value2;!This will lead to an error when value2 is
       equal to 0.
ERROR
  RAISE;
ENDPROC

ENDMODULE
```

V tomto příkladu bude dělení nulou mít za následek chybu. V `ERROR` handleru bude `RAISE` šířit chybu k chybovému handleru ve volající rutině "`main`". Stejné chybové číslo zůstane aktivní. `RETRY` provede znovu celou rutinu "`routine1`".

Vykonávání programu

Vykonávání programu pokračuje v chybovém handleru rutiny. Po provedení chybového handleru bude vykonávání programu pokračovat s:

- rutina, která volala zmíněnou rutinu (RETURN).
- chybový handler rutiny, která volala zmíněnou rutinu (RAISE).

Instrukce `RAISE` v chybovém handleru rutiny má také další funkci. Může se použít pro dlouhý skok (viz “Zotavení z chyby dlouhým skokem”). S dlouhým skokem je možné rozšířit chybu z chybového handleru, z hluboko vnořeného řetězce volání, na vyšší úroveň jedním krokem.

Jestliže instrukce `RAISE` je přítomna v trap rutině, chyba je ošetřena systémovým chybovým handlerem.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.184 RAISE - Volá chybový handler

RobotWare-OS

Pokračování

Řešení chyb

Jestliže chybové číslo je mimo rozsah, potom je nastavena systémová proměnná ERRNO na ERR_ILLRAISE (viz „Datové typy - errnum“). Tato chyba může být ošetřena v chybovém handleru.

Syntaxe

```
RAISE [<error number>] ';'
```

Související informace

Pro informace o	Viz
Řešení chyb	*Technická referenční příručka - Systémové parametry*
Zotavení z chyby s dlouhým skokem	*Technická referenční příručka - RAPID kernel*
Rezervování chybových čísel	*BookErrNo - Zapsat chybové číslo systému RAPID na str 41*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.185 RaiseToUser - Rozšiřuje chybu na uživatelskou úroveň

Použití

`RaiseToUser` se používá v chybovém handleru v rutinách nostepin k rozšíření aktuální chyby nebo každé jiné definované chyby k chybovému handleru na uživatelské úrovni. Uživatelská úroveň je v tomto případě první rutina v řetězci volání nad rutinou nostepin.

Základní příklady

Následující příklad názorně ukazuje instrukci `RaiseToUser`:

Příklad 1

```
MODULE MyModule
  VAR errnum ERR_MYDIVZERO:= -1;
  PROC main()
      BookErrNo ERR_MYDIVZERO;
      ...
      routine1;
      ...
      ERROR
      IF ERRNO = ERR_MYDIVZERO THEN
        TRYNEXT;
      ELSE
        RETRY;
      ENDIF
  ENDPROC
ENDMODULE
MODULE MySysModule (SYSMODULE, NOSTEPIN, VIEWONLY)
  PROC routine1()
    ...
    routine2;
    ...
  UNDO
  ! Free allocated resources
  ENDPROC
  PROC routine2()
    VAR num n:=0;
    ...
    reg1:=reg2/n;
    ...
    ERROR
    IF ERRNO = ERR_DIVZERO THEN
      RaiseToUser \Continue \ErrorNumber:=ERR_MYDIVZERO;
    ELSE
      RaiseToUser \BreakOff;
    ENDIF
  ENDPROC
ENDMODULE
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.185 RaiseToUser - Rozšiřuje chybu na uživatelskou úroveň

RobotWare - OS

Pokračování

Dělení nulou v `routine2` se bude šířit nahoru k chybovému handleru v main rutině s errno nastaveným na `ERR_MYDIVZERO`. Instrukce `TRYNEXT` v hlavním chybovém handleru potom způsobí pokračování vykonávání programu s instrukcí po dělení nulou v `routine2`. Přepínač \Continue kontroluje toto chování.

Jestliže jakékoliv jiné chyby se objeví v `routine2`, potom si přepínač \BreakOff vynutí pokračování vykonávání od chybového handleru v main rutině. V tomto případě bude proveden undo handler v routine1 za současného pozvednutí na uživatelskou úroveň. Instrukce RETRY v chybovém handleru v main rutině vykoná routine1 ještě jednou od začátku.

Undo handler v routine1 bude také vykonán v případě \Continue, jestliže následující `RAISE` nebo `RETURN` je proveden na uživatelské úrovni.

Argumenty

```
RaiseToUser[ \Continue] | [ \BreakOff][ \ErrorNumber]
```

`[\Continue]`

Datový typ: `switch`

Pokračovat ve vykonávání v rutině, která způsobila chybu.

`[\BreakOff]`

Datový typ:`switch`

Ukončit řetězec volání a pokračovat ve vykonávání na uživatelské úrovni. Každý undo handler v řetězci volání bude proveden odděleně od undo handleru v rutině, která pozvedla chybu.

Jeden z argumentů `\Continue` nebo `\BreakOff` musí být naprogramován, aby byla vyloučena chyba vykonávání.

`[\ErrorNumber]`

Datový typ: `errnum`

Jakékoliv číslo mezi 1 a 90, které může chybový handler použít k nalezení chyby, která se objevila (systémová proměnná `ERRNO`).

Je také možné rezervovat chybové číslo mimo rozsah 1-90 s instrukcí `BookErrNo`

Jestliže argument `\ErrorNumber` není určen, potom se původní chybové číslo rozšiřuje k chybovému handleru v rutině na uživatelské úrovni.

Vykonávání programu

`RaiseToUser` can může být použito pouze v chybovém handleru v rutině nostepin.

Vykonávání programu pokračuje v chybovém handleru rutiny na uživatelské úrovni. Chybové číslo zůstává aktivní, jestliže volitelný parametr `\ErrorNumber` není přítomen. Systémový chybový handler ošetří chybu, jestliže neexistuje chybový handler na uživatelské úrovni. Systémový chybový handler je volán, jestliže není určen žádný z argumentů `\Continue` nebo `\BreakOff`.

Existují dvě odlišná chování po provedení chybového handleru. Vykonávání programu pokračuje v rutině s `RaiseToUser`, jestliže je zapnut přepínač `\Continue`. Vykonávání programu pokračuje na uživatelské úrovni, jestliže je zapnut přepínač `\BreakOff`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu může pokračovat s:

- instrukcí, která způsobila chybu (RETRY)
- následující instrukcí (TRYNEXT)
- chybovým handlerem rutiny, která volala rutinu na uživatelské úrovni (RAISE)
- rutinou, která volala rutinu na uživatelské úrovni (RETURN)

Řešení chyb

Jestliže chybové číslo je mimo rozsah, potom je nastavena systémová proměnná ERRNO na ERR_ILLRAISE (viz „Datové typy - errnum“). Tato chyba bude ošetřena v systémovém chybovém handleru.

Syntaxe

```
RaiseToUser
  [ '\' Continue ]
  '|' [ '\' BreakOff ]
  [ '\' ErrorNumber ':=' ] < expression (IN) of errnum> ';'
```

Související informace

Pro informace o	Viz
Řešení chyb	*Technická referenční příručka - Přehled RAPID*
Zpracování undo	*Technická referenční příručka - Přehled RAPID*
Rezervování chybových čísel	*BookErrNo - Zapsat chybové číslo systému RAPID na str 41*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.186 ReadAnyBin - Přečíst data z binárního sériového kanálu nebo souboru

Použití

`ReadAnyBin` (*Read Any Binary*) se používá ke čtení každého typu dat z binárního sériového kanálu nebo souboru.

Základní příklady

Následující příklad názorně ukazuje instrukci `ReadAnyBin`:

Viz také *Další příklady na str 519*.

Příklad 1

```
VAR iodev channel1;
VAR robtarget next_target;
...
Open "com1:", channel1 \Bin;
ReadAnyBin channel1, next_target;
```

Další cíl robotu, který bude vykonán, `next_target`, je přečten z kanálu, na který odkazoval `channe12`.

Argumenty

```
ReadAnyBin IODevice Data [\Time]
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) binárního sériového kanálu nebo souboru, který bude načten.

`Data`

Datový typ: `ANYTYPE`

`VAR` nebo `PERS`, do kterého budou přečtená data uložena.

`[\Time]`

Datový typ: `num`

Max. čas pro čtecí operaci (vypršení času) v sekundách. Jestliže tento argument není určen, potom je max čas nastaven na 60 sekund. Má-li se čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Jestliže tento čas uběhne před dokončením čtecí operace, potom bude volán chybový handler s chybovým kódem `ERR_DEV_MAXTIME`. Jestliže neexistuje chybový handler, vykonávání bude zastaveno.

Funkce vypršení času se také používá během zastavení programu a bude uvedena programem RAPID při spouštění programu.

Vykonávání programu

Tolik bajtů, kolik je požadováno pro určená data, je načteno z určeného binárního sériového kanálu nebo souboru.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Více příkladů instrukce `ReadAnyBin` je názorně uvedeno dole.

Příklad 1

```
CONST num NEW_ROBT:=12;
CONST num NEW_WOBJ:=20;
VAR iodev channel;
VAR num input;
VAR robtarget cur_robt;
VAR wobjdata cur_wobj;

Open "com1:", channel\Bin;

! Wait for the opcode character
input := ReadBin (channel \Time:= 0.1);
TEST input
CASE NEW_ROBT:
  ReadAnyBin channel, cur_robt;
CASE NEW_WOBJ:
  ReadAnyBin channel, cur_wobj;
ENDTEST
Close channel;
```

Jako první krok je ze sériového kanálu načten `opcode` zprávy. Podle tohoto `opcode` je ze sériového kanálu načten `robtarget` nebo `wobjdata`.

Řešení chyb

Jestliže se objeví chyba během čtení, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`.

Jestliže vyprší čas před dokončením čtecí operace, potom je systémová proměnná `ERRNO` nastavena na `ERR_DEV_MAXTIME`.

Jestliže se objeví chyba kontrolního součtu během čtení dat, potom je systémová proměnná `ERRNO` nastavena na `ERR_RANYBIN_CHK`.

Jestliže je zjištěn konec souboru před přečtením všech bajtů, potom je systémová proměnná `ERRNO` nastavena na `ERR_RANYBIN_EOF`.

Tyto chyby mohou být potom ošetřeny chybovým handlerem.

Omezení

Tuto instrukci je možné používat pouze u sériových kanálů nebo souborů, které byly otevřeny pro binární čtení.

Data, která budou čtena touto instrukcí `ReadAnyBin`, musí být hodnotového datového typu jako `num`, `bool`, nebo `string`. Záznam, komponent záznamu, pole nebo prvek pole těchto hodnotových datových typů mohou být také použity. Celá

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.186 ReadAnyBin - Přečíst data z binárního sériového kanálu nebo souboru

RobotWare - OS

Pokračování

data nebo částečná data s polohodnotovými nebo nehodnotovými datovými typy se nemohou používat.

POZNÁMKA

Proměnná VAR nebo PERS pro ukládání přečtených dat může být aktualizována v několika krocích. Proto vždy čekejte na aktualizaci celé datové struktury, teprve potom použijte čtení dat (read data) z TRAP nebo jiné programové úlohy.

Protože WriteAnyBin-ReadAnyBin jsou určeny pouze k řešení interních binárních dat řadiče se sériovými kanály nebo soubory mezi nebo v rámci kontrolních systémů IRC5, žádný datový protokol není vydán a data nemohou být interpretována na jakémkoliv PC.

Vývoj ovládacího softwaru může přerušit kompatibilitu, a proto nemusí být možné používat WriteAnyBin-ReadAnyBin mezi různými verzemi softwaru RobotWare.

Syntaxe

```
ReadAnyBin
  [ IODevice ':='] <variable (VAR) of iodev> ','
  [ Data ':='] <var or pers (INOUT) of ANYTYPE>
  [ '\' Time ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Otevření sériových kanálů nebo souborů	*Technická referenční příručka - Přehled RAPID*
Zapsat data do binárního sériového kanálu nebo souboru	*WriteAnyBin - Zapisuje data do binárního sériového kanálu nebo souboru na str 983*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.187 ReadBlock - přečíst blok dat ze zařízení

Použití

`ReadBlock` se používá pro čtení bloku dat ze zařízení připojeného k sériovému rozhraní senzoru *. The data is stored in a file.* (Data jsou uložena do souboru)

Rozhraní senzoru komunikuje se dvěma senzory přes sériové kanály pomocí transportního protokolu RTP1.

Toto je příklad konfigurace kanálu senzoru.

COM_PHY_CHANNEL:

- Name “COM1:”
- Connector “COM1”
- Baudrate 19200

COM_TRP:

- Name “sen1:”
- Type “RTP1”
- PhyChannel “COM1”

Základní příklady

Následující příklad názorně ukazuje instrukci `ReadBlock`:

Příklad 1

```
CONST string SensorPar := "flp1:senpar.cfg";
CONST num ParBlock:= 1;

! Connect to the sensor device "sen1:" (defined in sio.cfg).
SenDevice "sen1:";

! Read sensor parameters from sensor datablock 1
! and store on flp1:senpar.cfg

ReadBlock "sen1:", ParBlock, SensorPar;
```

Argumenty

```
ReadBlock device BlockNo FileName [ \TaskName ]
```

`device`

Datový typ: string

Jméno I/O zařízení konfigurované v sio.cfg pro použitý senzor.

`BlockNo`

Datový typ: `num`

Argument `BlockNo` se používá pro výběr datového bloku v senzoru, který bude přečten.

`FileName`

Datový typ: `string`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.187 ReadBlock - přečíst blok dat ze zařízení

Sensor Interface

Pokračování

Argument `FileName` se používá pro definování souboru, do kterého budou zapsána data z datového bloku v senzoru zvoleného argumentem `BlockNo`.

`[ \TaskName ]`

Datový typ: `string`

Argument `TaskName` umožňuje přístup k zařízením v jiných úkolech RAPID.

Řešení chyb

Chybová konstanta (hodnota ERRNO)	Popis
SEN_NO_MEAS	Selhání měření
SEN_NOREADY	Senzor není schopen zpracovat příkaz
SEN_GENERRO	Obecná chyba senzoru
SEN_BUSY	Snímač je zaneprázdněn
SEN_UNKNOWN	Neznámý senzor
SEN_EXALARM	Externí chyba senzoru
SEN_CAALARM	Interní chyba senzoru
SEN_TEMP	Chyba teploty senzoru
SEN_VALUE	Neplatná hodnota komunikace
SEN_CAMCHECK	Selhání kontroly senzoru
SEN_TIMEOUT	Chyba komunikace

Syntaxe

```
ReadBlock
  [ device ':=' ] < expression(IN) of string> ','
  [ BlockNo ':=' ] < expression (IN) of num > ','
  [ FileName ':=' ] < expression (IN) of string > ','
  [ '\' TaskName ':=' < expression (IN) of string > ] ';'
```

Související informace

Pro informace o	Viz
Připojit k zařízení senzoru	*SenDevice - Připojit k zařízení senzoru na str 614*
Zapsat proměnnou senzoru	*WriteVar - Zapsat proměnnou na str 998*
Přečíst proměnnou senzoru	*ReadVar - Přečíst proměnnou ze zařízení na str 1297*
Zapsat datový blok senzoru	*WriteBlock - Zapsat blok dat do zařízení na str 988*
Konfigurace komunikace senzoru	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.188 ReadCfgData - Čte atribut systémového parametru

Použití

`ReadCfgData` se používá pro čtení jednoho atributu systémového parametru (konfigurační data).

Vedle čtení jmenovitých parametrů je také možné vyhledávat nejmenované parametry.

Základní příklady

Následující příklad ilustruje instrukci `ReadCfgData`. Oba tyto příklady ukazují, jak číst jmenovité parametry.

Příklad 1

```
VAR num offset1;
...
ReadCfgData "/MOC/MOTOR_CALIB/rob1_1","cal_offset",offset1;
```

Čte hodnotu ofsetu kalibrace pro osu 1 pro `rob_1` do `num` proměnné`offset1`.

Příklad 2

```
VAR string io_device;
...
ReadCfgData "/EIO/EIO_SIGNAL/process_error","Device",io_device;
```

Čte jméno I/O zařízení, kde je signál `process_error` definován do `string` proměnné `io_device`.

Argumenty

```
ReadCfgData InstancePath Attribute CfgData [\ListNo]
```

`InstancePath`

Datový typ: `string`

Určuje cestu k parametru, který má být získán.

Pro jmenovité parametry je formát tohoto řetězce `/DOMAIN/TYPE/ParameterName`.

Pro nejmenované parametry je formát tohoto řetězce `/DOMAIN/TYPE/Attribute/AttributeValue`.

`Attribute`

Datový typ: `string`

Jméno atributu parametru, který má být přečten.

`CfgData`

Datový typ: `any type`

Proměnná, kam bude uložena hodnota atributu. Podle typu atributu jsou platnými hodnotami `bool`, `num` nebo `string`.

`[\ListNo]`

Datový typ: `num`

Proměnná držící číslo instance `Attribute` + `AttributeValue`, které má být nalezeno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.188 ReadCfgData - Čte atribut systémového parametru

RobotWare - OS

Pokračování

První výskyt `Attribute` + `AttributeValue` má číslo instance 0. Jestliže se hledá více instancí, potom bude vrácená hodnota v `\ListNo` inkrementována s 1. Jinak, jestliže už neexistují žádné další instance, potom bude vrácená hodnota -1. Předdefinovanou konstantu `END_OF_LIST` je možné použít ke kontrole, jestli je možné hledat další instance.

Vykonávání programu

Hodnota atributu určeného argumentem `Attribute` je uložena do proměnné určené argumentem `CfgData`.

Při používání formátu `/DOMAIN/TYPE/ParameterName` v `InstancePath` mohou být získány pouze jmenovité parametry, tj. parametry, kde je prvním atributem `name`, `Name` nebo `NAME`.

U nejmenovaných parametrů použijte volitelný parametr `\ListNo` pro zvolení, ze které instance se bude číst hodnota atributu. Aktualizace k další dostupné instanci proběhne po každém úspěšném čtení.

Další příklady

Více příkladů instrukce `ReadCfgdata` je uvedeno dole. Oba tyto příklady ukazují, jak číst nejmenované parametry.

Příklad 1

```
VAR num list_index;
VAR string read_str;
...
list_index:=0;
ReadCfgData "/EIO/EIO_CROSS/Act1/do_13", "Res", read_str,
     \ListNo:=list_index;
IF read_str <> "" THEN
  TPWrite "Resultant signal for signal do_13 is: " + read_str;
ENDIF
```

Čte výsledný signál pro nejmenovaný digitální signál `di_13` a umisťuje jméno do `string` proměnné `read_str`.

V tomto příkladu má doména EIO následující cfg kód:

EIO_CROSS:

-Name "Cross_di_1_do_2" -Res "di_1" -Act1 "do_2"

-Name "Cross_di_2_do_2" -Res "di_2" -Act1 "do_2"

-Name "Cross_di_13_do_13" -Res "di_13" -Act1 "do_13"

Příklad 2

```
VAR num list_index;
VAR string read_str;
...
list_index:=0;
WHILE list_index <> END_OF_LIST DO
  read_str:="";
  ReadCfgData "/EIO/EIO_SIGNAL/Device/USERIO", "Name", read_str,
       \ListNo:=list_index;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  IF read_str <> "" THEN
    TPWrite "Signal: " + read_str;
  ENDIF
ENDWHILE
..
ERROR
  TRYNEXT;
```

Přečíst jména všech signálů definovaných pro I/O zařízení `USERIO`.

V tomto příkladu má doména EIO následující cfg kód:

```
EIO_SIGNAL:
  -Name "USERDO1" -SignalType "DO" -Device "USERIO" -DeviceMap "0"
  -Name "USERDO2" -SignalType "DO" -Device "USERIO" -DeviceMap "1"
  -Name "USERDO3" -SignalType "DO" -Device "USERIO" -DeviceMap "2"
```

Příklad 3

```
VAR num list_index;
VAR string read_str;
...
list_index:=0;
WHILE list_index <> END_OF_LIST DO
  read_str:="";
  ReadCfgData "/EIO/DEVICENET_DEVICE/Network/DeviceNet", "Name",
       read_str, \ListNo:=list_index;
  IF read_str <> "" THEN
    TPWrite read_str;
  ENDIF
ENDWHILE
..
ERROR
  TRYNEXT;
```

Přečíst jména všech zařízení DeviceNet.

V tomto příkladu má doména EIO následující cfg kód:

```
DEVICENET_DEVICE:
  -Name PANEL -Network "DeviceNet" -Simulated
  -Name DRV_1 -Network "DeviceNet" -Simulated
  -Name DEVICE1 -Network "DeviceNet" -Simulated
  -Name DEVICE2 -Network "DeviceNet" -Simulated
```

Řešení chyb

Jestliže není možné najít data určená s “`InstancePath` + `Attribute`” v konfigurační databázi, potom se systémová proměnná `ERRNO` nastaví na `ERR_CFG_NOTFND`.

Jestliže datový typ pro parametr `CfgData` není totožný se skutečným datovým typem pro nalezená data určená s “`InstancePath` + `Attribute`” v konfigurační databázi, potom se systémová proměnná `ERRNO` nastaví na `ERR_CFG_ILLTYPE`.

Při pokusu o čtení interních dat se potom systémová proměnná `ERRNO` nastaví na `ERR_CFG_INTERNAL`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.188 ReadCfgData - Čte atribut systémového parametru
RobotWare - OS
Pokračování

Jestliže proměnná v argumentu \ListNo má při vykonávání instrukce hodnotu mimo rozsah dostupných instancí (0 ... n), potom se ERRNO nastaví na ERR_CFG_OUTOFBOUNDS.

Tyto chyby mohou být zpracovány v chybovém handleru.

Omezení

Konverzi z jednotek systémového parametru (m, radián, sekunda atd.) na jednotky programu RAPID (mm, stupeň, sekunda atd.) pro CfgData datového typu num musí provést uživatel v programu RAPID.

Při používání formátu /DOMAIN/TYPE/ParameterName v InstancePath mohou být získány pouze jmenovité parametry, tj. parametry, kde je prvním atributem name, Name nebo NAME.

Řetězce RAPID jsou omezeny na 80 znaků. V některých případech to může být teoreticky příliš málo pro definici InstancePath, Attribute nebo CfgData.

Předdefinovaná data

Předdefinovaná konstanta END_OF_LIST s hodnotou -1 se může použít k ukončení čtení, když není možné nalézt další instance.

Syntaxe

```
ReadCfgData
  [ InstancePath ':=' ] < expression (IN) of string > ','
  [ Attribute ':=' ] < expression (IN) of string > ','
  [ CfgData ':=' ] < variable (VAR) of anytype >
  [ '\' ListNo ':=' < variable (VAR) of num >] ';'
```

Související informace

Pro informace o	Viz
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Zapsat atribut systémového parametru	*WriteCfgData - Zapisuje atribut systémového parametru na str 990*
Získat jméno robotu v aktuální úloze	*RobName - Získat jméno TCP robotu na str 1304*
Konfigurace	*Technická referenční příručka - Systémové parametry*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.189 ReadErrData - Získává informace o chybě

Použití

`ReadErrData` se použije v trap rutině, aby byly získány informace (doména, typ, číslo a prolínající se %s řetězců) o chybě, změně stavu nebo varování, které způsobily vykonání trap rutiny.

Základní příklady

Následující příklad názorně ukazuje instrukci `ReadErrData`:

Viz kapitola *Další příklady na str 528*.

Příklad 1

```
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
VAR trapdata err_data;
VAR string titlestr;
VAR string string1;
VAR string string2;
...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number,
  err_type \Title:=titlestr \Str1:=string1 \Str2:=string2;
ENDTRAP
```

Jestliže je chyba zachycena do trap rutiny `trap_err`, doména chyby, číslo chyby, typ chyby a dva první prolínající se řetězce v chybové zprávě jsou uloženy do příslušných proměnných.

Argumenty

```
ReadErrData TrapEvent ErrorDomain ErrorId ErrorType [\Title] [\Str1]
     [\Str2] [\Str3] [\Str4] [\Str5]
```

`TrapEvent`

Datový typ: `trapdata`

Proměnná obsahující informace o tom, co způsobilo vykonání trapu.

`ErrorDomain`

Datový typ: `errdomain`

Proměnná pro uložení chybové domény, do které patří chyba, změna stavu nebo varování, které se objevily. Ref. k předdefinovaným datům typu `errdomain`.

`ErrorId`

Datový typ: `num`

Proměnná k uložení čísla chyby, která se objevila. Číslo chyby je vráceno bez první číslice (chybová doména) a bez počátečních nul kompletního chybového čísla.

Např. 10008. Restartovaný program je vrácen jako 8.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.189 ReadErrData - Získává informace o chybě

RobotWare - OS

Pokračování

`ErrorType`

Datový typ: `errtype`

Proměnná pro uložení typu události, jako je chyba, změna stavu nebo varování, které se objevily. Ref. k předdefinovaným datům typu `errtype`**.**

`[ \Title ]`

Datový typ: `string`

Proměnná k uložení titulu chybové zprávy. Titul je ve formátu UTF8 a všechny znaky nebudou zobrazeny správně pro všechny jazyky na FlexPendantu.

`[ \Str1 ] ... [ \Str5 ]`

Datový typ: `string`

Aktualizovat proměnnou určeného řetězce s argumentem, který se prolíná v chybové zprávě. Může existovat až pět argumentů ve zprávě typu %s, %f, %d nebo %ld, což bude vždy převedeno na řetězec při vykonávání této instrukce. `Str1` **bude držet první argument,** `Str2` **bude držet druhý argument atd. Informace o tom, kolik argumentů je ve zprávě, můžeme nalézt v *Provozní příručka - Odstraňování závad*. Prolínající se argumenty jsou označeny v tomto dokumentu jako arg.**

Vykonávání programu

Proměnné `ErrorDomain`, `ErrorId`, `ErrorType`, `Title` **a** `Str1 ... Str5` **se aktualizují podle obsahu** `TrapEvent`.

Jestliže různé události jsou připojeny ke stejné trap rutině, potom program musí zajistit, aby událost měla souvislost se sledováním chyb. Toho může být docíleno testováním, jestli `INTNO` **souhlasí s číslem přerušení použitým v instrukci** `IError`;

Další příklady

Více příkladů instrukce `ReadErrData` **je názorně uvedeno dole.**

Příklad 1

```
VAR intnum err_interrupt;
VAR trapdata err_data;
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
...
PROC main()
  CONNECT err_interrupt WITH trap_err;
  IError COMMON_ERR, TYPE_ERR, err_interrupt;
  ...
  IDelete err_interrupt;
  ...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
  ! Set domain no 1 ... 11
  SetGO go_err1, err_domain;
  ! Set error no 1 ...9999
  SetGO go_err2, err_number;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
ENDTRAP
```

Když se objeví chyba (pouze chyby, nikoliv varování nebo změna stavu) číslo chyby je vyhledáno v trap rutině a tato hodnota se použije k nastavení 2 skupin digitálních výstupních signálů.

Omezení

Není možné získat informace o interních chybách.

Syntaxe

```
ReadErrData
  [TrapEvent ':='] <variable (VAR) of trapdata>',’
  [ErrorDomain' :='] <variable (VAR) of errdomain>',’
  [ErrorId':='] <variable (VAR) of num>',’
  [ErrorType' :='] <variable (VAR) of errtype>
  ['\'Title ´:='<variable (VAR) of string>]
  ['\'Str1 ´:='<variable (VAR) of string>]
  ['\'Str2 ´:='<variable (VAR) of string>]
  ['\'Str3 ´:='<variable (VAR) of string>]
  ['\'Str4 ´:='<variable (VAR) of string>]
  ['\'Str5 ´:='<variable (VAR) of string>]';’
```

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*
Více informací o správě přerušení	*Technická referenční příručka - Přehled RAPID*
Chybové domény, předdefinované konstanty	*errdomain - Chybová doména na str 1493*
Typy chyb, předdefinované konstanty	*errtype - Typ chyby na str 1503*
Přikazuje přerušení na chyby	*IError - Přikazuje přerušení na chyby na str 246*
Získat data přerušení pro aktuální TRAP	*GetTrapData - Získat data přerušení pro aktuální TRAP na str 230*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.190 ReadRawBytes - Přečíst data rawbytes

Použití

`ReadRawBytes` se používá pro čtení dat typu `rawbytes` ze zařízené otevřeného s `Open\Bin`.

Základní příklady

Následující příklad názorně ukazuje instrukci `ReadRawBytes`:

Příklad 1

```
VAR iodev io_device;
VAR rawbytes raw_data_out;
VAR rawbytes raw_data_in;
VAR num float := 0.2;
VAR string answer;

ClearRawBytes raw_data_out;
PackDNHeader "10", "20 1D 24 01 30 64", raw_data_out;
PackRawBytes float, raw_data_out, (RawBytesLen(raw_data_out)+1)
     \Float4;

Open "/FC1:/dsqc328_1", io_device \Bin;
WriteRawBytes io_device, raw_data_out;
ReadRawBytes io_device, raw_data_in \Time:=1;
Close io_device;
UnpackRawBytes raw_data_in, 1, answer \ASCII:=10;
```

V tomto příkladu je `raw_data_out` vyčištěno a potom zabaleno s hlavičkou DeviceNet a float s hodnotou 0.2.

Zařízení `"/FC1:/dsqc328_1"` je otevřeno a aktuálně platná data v `raw_data_out` jsou zapsána do zařízení. Potom program čeká nejdéle `1` sek. na čtení ze zařízení, které je uloženo do `raw_data_in`.

Po zavření zařízení `"/FC1:/dsqc328_1"` jsou přečtená data rozbalena jako řetězec znaků a uložena do `answer`.

Argumenty

```
ReadRawBytes IODevice RawData [\Time]
```

`IODevice`

Datový typ: `iodev`

`IODevice` je identifikátor zařízení, ze kterého budou přečtena data.

`RawData`

Datový typ: `rawbytes`

`RawData` je datový kontejner, který obsahuje přečtená data z `IODevice` a začíná na indexu 1.

`[\Time]`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Max. čas pro čtecí operaci (vypršení času) v sekundách (rozlišení 0,001 s). Jestliže tento argument není určen, potom je max. čas nastaven na 60 sekund. Má-li se čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Jestliže tento čas uběhne před dokončením čtecí operace, potom bude volán chybový handler s chybovým kódem `ERR_DEV_MAXTIME`. Jestliže neexistuje chybový handler, vykonávání bude zastaveno.

Funkce vypršení času se také používá během zastavení programu a bude uvedena programem RAPID při spouštění programu.

Vykonávání programu

Během vykonávání programu jsou data čtena ze zařízení označeného `IODevice`.

Při používání `WriteRawBytes` pro příkazy aplikační sběrnice, jako je DeviceNet, potom aplikační sběrnice vždy odesílá odpověď. Odpověď musí být zpracována v RAPIDu s instrukcí `ReadRawBytes`.

Aktuální délka platných bajtů v proměnné `RawData` je nastavena na čtení řady bajtů. Data začínají na indexu 1 v `RawData`.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Řešení chyb

Jestliže se objeví chyba během čtení, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`.

Jestliže vyprší nastavený čas před čtecí operací, potom není ovlivněno nic v proměnné `RawData` a systémová proměnná `ERRNO` se nastaví na `ERR_DEV_MAXTIME`.

Tyto chyby mohou být potom ošetřeny chybovým handlerem.

Syntaxe

```
ReadRawBytes
  [IODevice ':=' ] < variable (VAR) of iodev>' ,'
  [RawData ':=' ] < variable (VAR) of rawbytes> ','
  [ '\' Time ':=' < expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Data`rawbytes`	*rawbytes - Data raw na str 1559*
Získat délku dat `rawbytes`	*RawBytesLen - Získat délku dat rawbytes na str 1278*
Vyčistit obsah dat `rawbytes`	*ClearRawBytes - Vyčistit obsahy rawbyte dat na str 118*
Kopírovat obsah dat `rawbytes`	*CopyRawBytes - Kopírovat obsah dat rawbytes na str 137*
Zabalit hlavičku DeviceNet do dat `rawbytes`	*PackDNHeader - Zabalit hlavičku DeviceNet do dat rawbytes na str 446*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.190 ReadRawBytes - Přečíst data rawbytes

RobotWare - OS

Pokračování

Pro informace o	Viz
Zabalit data do dat `rawbytes`	*PackRawBytes - Zabalit data do dat rawbytes na str 449*
Zapsat data `rawbytes`	*WriteRawBytes - Zapsat data rawbytes na str 994*
Rozbalit data z dat `rawbytes`	*UnpackRawBytes - Rozbalit data z rawbyte dat na str 908*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.191 RemoveAllCyclicBool - Odstranit všechny cyklicky hodnocené logické podmínky

Použití

`RemoveAllCyclicBool` se používá k odstranění cyklického hodnocení všech logických podmínek.

Základní příklady

Následující příklad názorně ukazuje instrukci `RemoveAllCyclicBool`.

Příklad 1

```
PERS bool cyclicflag1;
TASK PERS bool cyclicflag2;
PERS bool mypersbool:=FALSE;

PROC main()
  SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
  SetupCyclicBool cyclicflag2, di3 AND di4 AND mypersbool=FALSE;
  ...
  RemoveAllCyclicBool;
  ...
```

První dvě cyklická hodnocení jsou nastavena, později je odstraněno cyklické hodnocení všech logických podmínek.

Syntaxe

```
RemoveAllCyclicBool ';'
```

Související informace

Pro informace o	Viz
Nastavit cyklicky hodnocenou logickou podmínku	*SetupCyclicBool - Nastavit cyklicky hodnocenou logickou podmínku na str 636*
Odstranit cyklicky hodnocenou logickou podmínku	*RemoveCyclicBool - Odstranit cyklicky hodnocenou logickou podmínku na str 534*
Cyklicky hodnocené logické podmínky, *Cyclic bool*.	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.192 RemoveCyclicBool - Odstranit cyklicky hodnocenou logickou podmínku

Použití

`RemoveCyclicBool` se používá k odstranění cyklického hodnocení konkrétní logické podmínky.

Základní příklady

Následující příklad názorně ukazuje instrukci `RemoveCyclicBool`.

Příklad 1

```
PERS bool cyclicflag1;

PROC main()
SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
...
RemoveCyclicBool cyclicflag1;
...
```

Nejprve je nastaveno cyklické hodnocení, později je odstraněno.

Argumenty

```
RemoveCyclicBool Flag
```

`Flag`

Datový typ: `bool`

Booleovská proměnná perzistentu, která obsahuje uloženou hodnotu logické podmínky.

Syntaxe

```
RemoveCyclicBool
  [ Flag ':=' ] <persistent (PERS) of bool> ';'
```

Související informace

Pro informace o	Viz
Nastavit cyklicky hodnocenou logickou podmínku	*SetupCyclicBool - Nastavit cyklicky hodnocenou logickou podmínku na str 636*
Odstranit všechny cyklicky hodnocené logické podmínky	*RemoveAllCyclicBool - Odstranit všechny cyklicky hodnocené logické podmínky na str 533*
Cyklicky hodnocené logické podmínky, *Cyclic bool*.	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.193 RemoveDir - Vymazat adresář

Použití

`RemoveDir` se používá pro odstranění adresáře.

Uživatel musí mít oprávnění pro zápis a provádění adresáře a adresář musí být prázdný.

Základní příklady

Následující příklad názorně ukazuje instrukci `RemoveDir`:

Příklad 1

```
RemoveDir "HOME:/mydir";
```

V tomto příkladu je adresář `mydir` pod `HOME:` vymazán.

Argumenty

```
RemoveDir Path
```

`Path`

Datový typ: `string`

Jméno adresáře k vymazání, určeného s plnou nebo relativní cestou.

Řešení chyb

Jestliže adresář neexistuje nebo adresář není prázdný nebo uživatel nemá oprávnění zápisu a provádění ke knihovně, potom se systémová proměnná `ERRNO` nastaví na `ERR_FILEACC`. Tuto chybu je potom možné ošetřit v chybovém handleru.

Syntaxe

```
RemoveDir
[ Path':=' ] < expression (IN) of string>';'
```

Související informace

Pro informace o	Viz
Adresář	*dir - Struktura adresáře souborů na str 1484*
Otevřít adresář	*OpenDir - Otevřít adresář na str 444*
Načíst adresář	*ReadDir - Přečíst další vstupní data v adresáři na str 1283*
Zavřít adresář	*CloseDir - Zavřít adresář na str 125*
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*
Kopírovat soubor	*CopyFile - Kopírovat soubor na str 135*
Zkontrolovat typ souboru	*IsFile - Zkontrolovat typ souboru na str 1207*
Zkontrolujte velikost souboru	*FileSize - Vyhledat velikost souboru na str 1155*
Zkontrolujte velikost souborového systému	*FSSize - Vyhledat velikost souborového systému na str 1161*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.193 RemoveDir - Vymazat adresář

RobotWare - OS

Pokračování

Pro informace o	Viz
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.194 RemoveFile - Vymazat soubor

Použití

`RemoveFile` se používá k odstranění souboru. Uživatel musí mít oprávnění zápisu a provádění k adresáři, kde je soubor, a uživatel musí mít oprávnění zápisu k samotnému souboru.

Základní příklady

Následující příklad názorně ukazuje instrukci `RemoveFile`:

Příklad 1

```
RemoveFile "HOME:/mydir/myfile.log";
```

V tomto příkladu je adresář `myfile.log` v adresáři `mydir` na disku `HOME:` vymazán.

Argumenty

```
RemoveFile Path
```

`Path`

Datový typ: string (řetězec)

Jméno souboru k vymazání, určeného s plnou nebo relativní cestou.

Řešení chyb

Jestliže soubor neexistuje, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
RemoveFile
[ Path':=' ] < expression (IN) of string>';'
```

Související informace

Pro informace o	Viz
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Kopírovat soubor	*CopyFile - Kopírovat soubor na str 135*
Zkontrolovat typ souboru	*IsFile - Zkontrolovat typ souboru na str 1207*
Zkontrolujte velikost souboru	*FileSize - Vyhledat velikost souboru na str 1155*
Zkontrolujte velikost souborového systému	*FSSize - Vyhledat velikost souborového systému na str 1161*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.195 RemoveSuperv - Odstranit podmínku pro jeden signál

Použití

`RemoveSuperv` se používá pro odstranění podmínek, které přidal `SetupSuperv` z dohledu.

Základní příklad

```
PROC main()
  InitSuperv;
  SetupSuperv diWR_EST, ACT, SUPERV_MAIN \ErrIndSig:= do_WR_Sup;
  SetupSuperv diGA_EST, ACT, SUPERV_MAIN;
  CapL p2, v100, cdata1, weavestart, weave,fine, tWeldGun;
  RemoveSuperv di_Arc_Sup, ACT, SUPERV_START;
ENDPROC
```

Odstraňuje signál *di_Arc_Sup* ze seznamu START.

Argumenty

```
RemoveSuperv Signal Condition Listtype
```

Signal

Datový typ: `signaldi`

Digitální signál, který bude odstraněn ze seznamu dohledu.

Condition

Datový typ: `num`

Jméno představující jednu z následujících dostupných podmínek:

ACT:	Používá se pro dohled nad statutem. Očekávaný status signálu během dohledu: aktivní. Jestliže se signál stane pasivním, dohled se spustí.
PAS:	Používá se pro dohled nad statutem. Očekávaný status signálu během dohledu: pasivní. Jestliže se signál stane aktivním, dohled se spustí.
POS_EDGE:	Používá se pro adresování dohledu. Očekávaný status signálu na konci dohledu: aktivní. Jestliže se signál nestane aktivním ve zvoleném časovém úseku, spustí se dohled.
NEG_EDGE:	Používá se pro dohled nad navazováním spojení. Očekávaný status signálu na konci dohledu: aktivní. Jestliže se signál nestane aktivním ve zvoleném časovém úseku, spustí se dohled.

Listtype

Datový typ: `num`

Jméno reprezentující počet různých seznamů (například fáze v procesu):

- SUPERV_PRE
- SUPERV_PRE_START
- SUPERV_END_PRE
- SUPERV_START
- SUPERV_MAIN
- SUPERV_END_MAIN

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

- SUPERV_START_POST1
- SUPERV_POST1
- SUPERV_END_POST1
- SUPERV_START_POST2
- SUPERV_POST2
- SUPERV_END_POST2

Syntaxe

```
RemoveSuperv
  [Signal ':='] < variable (VAR) of signaldi > ','
  [Condition ':='] < variable (IN) of num > ','
  [Listtype ':='] < variable (IN) of num >';'
```

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instrukce `InitSuperv`	*InitSuperv - Resetovat veškerý dohled pro CAP na str 271*
Instrukce `SetupSuperv`	*SetupSuperv - Nastavit podmínky pro dohled signálu v CAP na str 639*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.196 RenameFile - Přejmenovat soubor

Použití

`RenameFile` se používá pro určení nového jména existujícímu souboru. Může se také používat pro přesun souboru z jednoho místa na druhé ve struktuře adresáře.

Základní příklady

Následující příklad názorně ukazuje instrukci `RenameFile`:

Příklad 1

```
RenameFile "HOME:/myfile", "HOME:/yourfile;
```

Souboru `myfile` je určeno jméno `yourfile`.

```
RenameFile "HOME:/myfile", "HOME:/mydir/yourfile";
```

Souboru `myfile` je určeno jméno `yourfile` a je přesunut do adresáře `mydir`.

Argumenty

```
RenameFile OldPath NewPath
```

`OldPath`

Datový typ:`string`

Kompletní cesta souboru, který bude přejmenován.

`NewPath`

Datový typ:`string`

Kompletní cesta přejmenovaného souboru.

Vykonávání programu

Souboru určenému v `OldPath` bude dáno jméno určené v `NewPath`. Jestli cesta v `NewPath` je odlišná od cesty v `OldPath`, potom bude soubor také přemístěn na nové místo.

Řešení chyb

Jestliže soubor určený v `NewPath` již existuje, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEEXIST`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
RenameFile
[ OldPath' :=' ] < expression (IN) of string > ','
[ NewPath' :=' ] < expression (IN) of string >';'
```

Související informace

Pro informace o	Viz
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Kopírovat soubor	*CopyFile - Kopírovat soubor na str 135*
Zkontrolovat typ souboru	*IsFile - Zkontrolovat typ souboru na str 1207*
Zkontrolujte velikost souboru	*FileSize - Vyhledat velikost souboru na str 1155*
Zkontrolujte velikost souborového systému	*FSSize - Vyhledat velikost souborového systému na str 1161*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.197 Reset - Resetuje digitální výstupní signál

Použití

`Reset` se používá pro resetování hodnoty digitálního výstupního signálu na nulu.

Základní příklady

Následující příklady názorně ukazují instrukci `Reset`:

Příklad 1

```
Reset do15;
```

Signál `do15` je nastaven na nulu.

Příklad 2

```
Reset weld;
```

Signál `weld` je nastaven na nulu.

Argumenty

```
Reset Signal
```

`Signal`

Datový typ: `signaldo`

Jméno signálu, který bude resetován na nulu.

Vykonávání programu

Absolutní (True) hodnota závisí na konfiguraci signálu. Jestliže je signál invertován v systémových parametrech, potom tato instrukce zajistí nastavení fyzického kanálu na 1.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v chybovém handleru. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
Reset
  [ Signal ':=' ] < variable (VAR) of signaldo > ';'
```

Související informace

Pro informace o	Viz
Nastavení digitálního výstupního signálu	*Set - Nastavuje digitální výstupní signál na str 616*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.198 ResetPPMoved - Resetovat stav pro ukazatel programu posunutý v ručním režimu

Použití

`ResetPPMoved` stav resetu pro ukazatel programu posunutý v ručním režimu. `PPMovedInManMode` vrací TRUE, jestliže uživatel posunul ukazatel programu, zatímco řadič je v ručním režimu - to znamená, že klíč operátora je na Ruční omezené rychlosti nebo Ruční plné rychlosti. Stav posunutého ukazatele programu se resetuje, když klíč je přepnut z Auto na Man nebo když se použije instrukce `ResetPPMoved`.

Základní příklady

Následující příklad názorně ukazuje instrukci `ResetPPMoved`:

Příklad 1

```
IF PPMovedInManMode() THEN
  WarnUserOfPPMovement;
  ! DO THIS ONLY ONCE
  ResetPPMoved;
  DoJob;
ELSE
  DoJob;
ENDIF
```

Vykonávání programu

Resetuje stav pro ukazatel programu posunutý v ručním režimu u aktuální programové úlohy.

Syntaxe

```
ResetPPMoved';'
```

Související informace

Pro informace o	Viz
Otestovat, jestli byl ukazatel programu posunut v ručním režimu	*PPMovedInManMode - Otestovat, jestli se ukazatel programu posunul v ručním režimu na str 1273*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.199 ResetRetryCount - Resetovat počet pokusů

Použití

`ResetRetryCount` se používá pro restování počtu nových pokusů, které provedl chybový handler. Max. počet pokusů, které mohou být provedeny, je definováno v konfiguraci.

Základní příklady

Následující příklad názorně ukazuje instrukci `ResetRetryCount`:

Příklad 1

```
VAR num myretries := 0;
...
ERROR
  IF myretries > 2 THEN
    ResetRetryCount;
    myretries := 0;
    TRYNEXT;
  ENDIF
  myretries:= myretries + 1;
  RETRY;
...
```

Tento programu bude opakovat problémovou instrukci 3x a potom zkusí další instrukci. Interní systémové počítadlo nových pokusů se resetuje před pokusem o novou instrukci (i když je toto provedeno systémem na `TRYNEXT`).

Vykonávání programu

U každého RETRY (nový pokus) provedeného chybovým handlerem bude interní systémové počítadlo kontrolovat, jestli nebyl překročen max počet nových pokusů určených v konfiguraci. Vykonávání instrukce `ResetRetryCount` bude resetovat počítadlo a umožní znovu provést max počet nových pokusů.

Syntaxe

```
ResetRetryCount ';'
```

Související informace

Pro informace o	Viz
Obslužné programy pro řešení chyb	*Technická referenční příručka - Přehled RAPID*
Obnovit vykonávání po chybě	*RETRY - Obnovit vykonávání po chybě na str 548*
Konfigurovat max počet nových pokusů	*Technická referenční příručka - Systémové parametry*
Počet zbývajících nových pokusů	*RemainingRetries - Zbývající nové pokusy, které se mají udělat na str 1301*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.200 RestoPath - Obnovuje cestu po přerušení

Použití

`RestoPath` se používá k obnovení cesty, která byla uložena v předchozí fázi pomocí instrukce `StorePath`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `RestoPath`:

Viz také *Další příklady* dole.

Příklad 1

```
RestoPath;
```

Obnovuje cestu, která byla uložena dříve pomocí `StorePath`.

Vykonávání programu

Aktuální dráha pohybu robotu a externí osy je vymazána a cesta uložená dříve pomocí `StorePath` je obnovena. Všimněte si, že nedochází k žádnému pohybu, dokud není vykonána instrukce `StartMove` nebo dokud není proveden návrat pomocí `RETRY` od chybového handleru.

Další příklady

Více příkladů jak používat instrukci `RestoPath` je názorně uvedeno dole.

Příklad 1

```
ArcL p100, v100, seam1, weld5 \Weave:=weave1, z10, gun1;
...
ERROR
  IF ERRNO=AW_WELD_ERR THEN
    gun_cleaning;
    StartMoveRetry;
  ENDIF
...
PROC gun_cleaning()
  VAR robtarget p1;
  StorePath;
  p1 := CRobT();
  MoveL pclean, v100, fine, gun1;
  ...
  MoveL p1, v100, fine, gun1;
  RestoPath;
ENDPROC
```

Při události chyby svařování pokračuje vykonávání programu v chybovém handleru rutiny, která volá `gun_cleaning`. Dráha pohybu, která je v té době vykonávána, je potom uložena a robot se přesune do pozice `pclean`, kde byla chyba napravena. Po dokončení se robot vrací na pozici, kde vznikla chyba, `p1`, a ukládá ještě jednou původní pohyb. Svařování se potom automaticky znovu spustí, což znamená, že

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

robot je nejprve reverzován podél dráhy, než začne svařování, a obvyklé vykonávání programu může pokračovat.

Omezení

Pouze data dráhy pohybu se ukládají s instrukcí `StorePath`. Jestliže uživatel chce přikázat pohyby na úrovni nové dráhy, potom musí být uložena aktuální stop pozice přímo po `StorePath` a předtím, než `RestoPath` provede pohyb k uložené stop pozici na dráze.

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (zonedata `fine`), nikoliv s průjezdným bodem, jinak nebude možný restart po selhání napájení.

`RestoPath` se nemůže provádět v RAPID rutině připojené ke kterékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart nebo Step.

Syntaxe

```
RestoPath´;´
```

Související informace

Pro informace o	Viz
Ukládání drah	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742*
Další příklady	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742* *PathRecStart - Spustit záznamník dráhy na str 467* *SyncMoveSuspend - Nastavit nezávislé-polokoordinované pohyby na str 766*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.201 RETRY - Obnovit vykonávání po chybě

Použití

Instrukce RETRY se používá po obnovení vykonávání programu po chybě a začíná novým provedením instrukce, která způsobila chybu.

Základní příklady

Následující příklad názorně ukazuje instrukci RETRY:

Příklad 1

```
reg2 := reg3/reg4;
...
ERROR
  IF ERRNO = ERR_DIVZERO THEN
    reg4 :=1;
    RETRY;
  ENDIF
```

Je proveden pokus o rozdělení reg3 a reg4. Jestliže reg4 je rovný 0 (dělení nulou), potom je proveden skok k chybovému handleru, který inicializuje reg4. Instrukce RETRY se potom použije ke skoku od chybového handleru a je proveden další pokus o dokončení rozdělení.

Vykonávání programu

Vykonávání programu pokračuje (obnovené vykonávání) s instrukcí, která způsobila chybu.

Řešení chyb

Jestliže je překročen max počet nových pokusů (4 nové pokusy), potom se vykonávání programu zastaví s chybovou zprávou. Max počet nových pokusů může být konfigurován v systémových parametrech (*System Misc*).

Omezení

Instrukce může existovat pouze v chybovém handleru rutiny. Jestliže chyba byla vytvořena pomocí instrukce RAISE, potom vykonávání programu nemůže být znovu spuštěno s instrukcí RETRY. Potom by se měla použít funkce TRYNEXT.

Syntaxe

```
RETRY ';'
```

Související informace

Pro informace o	Viz
Obslužné programy pro řešení chyb	*Technická referenční příručka - Přehled RAPID*
Konfigurovat max počet nových pokusů	*Technická referenční příručka - Systémové parametry*
Pokračovat s další instrukcí	*TRYNEXT - Přeskočí instrukci, která způsobila chybu na str 884*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.202 RETURN - Ukončení vykonávání rutiny

Použití

RETURN se používá k ukončení vykonávání rutiny. Jestliže rutina je funkcí, potom je vrácena také hodnota funkce.

Základní příklady

Následující příklady názorně ukazují instrukci RETURN:

Příklad 1

```
errormessage;
Set do1;
...
PROC errormessage()
  IF di1=1 THEN
    RETURN;
  ENDIF
  TPWrite "Error";
ENDPROC
```

Procedura errormessage je volána. Jestliže tato procedura dojde k instrukci RETURN, vykonávání programu se vrací k instrukci následující po volání procedury, Set do 1.

Příklad 2

```
FUNC num abs_value(num value)
  IF value<0 THEN
    RETURN -value;
  ELSE
    RETURN value;
  ENDIF
ENDFUNC
```

Funkce vrací absolutní hodnotu čísla.

Argumenty

```
RETURN [ Return value ]
```

Return value

Datový typ: Podle deklarace funkce.

Vratná hodnota funkce.

Vratná hodnota musí být určena v instrukci RETURN přítomné ve funkci.

Jestliže instrukce je přítomna v proceduře nebo trap rutině, potom by vratná hodnota neměla být stanovována.

Vykonávání programu

Výsledek instrukce RETURN se může lišit podle typu rutiny, ve které je použita:

- Hlavní (Main) rutina: Jestliže program má jednoduchý cyklus režimu běhu, potom se program zastaví. Jinak vykonávání programu pokračuje s první instrukcí hlavní rutiny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.202 RETURN - Ukončení vykonávání rutiny

RobotWare - OS

Pokračování

- Procedura: Vykonávání programu pokračuje s instrukcí následující po volání procedury.
- Funkce: Vrací hodnotu funkce.
- Trap rutina: Vykonávání programu pokračuje od místa, kde vzniklo přerušení.
- Chybový handler v proceduře: Vykonávání programu pokračuje s rutinou, která volala rutinu s chybovým handlerem (s instrukcí následující po volání procedury).
- Chybový handler ve funkci: Hodnota funkce je vrácena.

Syntaxe

```
RETURN [ <expression> ]';'
```

Související informace

Pro informace o	Viz
Funkce a procedury	*Technická referenční příručka - Přehled RAPID*
Trap rutiny	*Technická referenční příručka - Přehled RAPID*
Obslužné programy pro řešení chyb	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.203 Rewind - Vrácení pozice souboru na začátek

Použití

`Rewind`nastavuje pozici souboru na začátek souboru.

Základní příklady

Následující příklad názorně ukazuje instrukci `Rewind`:

Viz také *Další příklady na str 551*.

Příklad 1

```
Rewind iodev1;
```

Soubor, na který odkazuje `iodev1` bude mít pozici souboru nastavenu na začátek souboru.

Argumenty

```
Rewind IODevice
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) souboru, který bude vrácen na začátek.

Vykonávání programu

Určený soubor je vrácen na začátek.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Další příklady

Více příkladů instrukce `Rewind` je názorně uvedeno dole.

Příklad 1

```
! IO device and numeric variable for use together with a binary
! file
VAR iodev dev;
VAR num bindata;

! Open the binary file with \Write switch to erase old contents
Open "HOME:"\File := "bin_file",dev \Write;
Close dev;

! Open the binary file with \Bin switch for binary read and write
! acess
Open "HOME:"\File := "bin_file",dev \Bin;
WriteStrBin dev,"Hello world";

! Rewind the file pointer to the beginning of the binary file
! Read contents of the file and write the binary result on TP
! (gives 72 101 108 108 111 32 119 111 114 108 100 )
Rewind dev;
bindata := ReadBin(dev);
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.203 Rewind - Vrácení pozice souboru na začátek

RobotWare - OS

Pokračování

```
WHILE bindata <> EOF_BIN DO
  TPWrite " " \Num:=bindata; bindata := ReadBin(dev);
ENDWHILE

! Close the binary file
Close dev;
```

Instrukce `Rewind` se používá pro vrácení binárního souboru na začátek tak, že obsah souboru může být přečten zpět s `ReadBin`

Omezení

U`Virtual Controller` neexistuje omezení, jestliže použitý soubor byl otevřen s přepínačem `\Bin` nebo `\Bin \Append`, `Rewind` před jakýmkoliv typem instrukce`Write` bude neúčinný. Zápis bude proveden na konec souboru.

Řešení chyb

Jestliže se během vracení na začátek objeví chyba, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
Rewind [IODevice ':='] <variable (VAR) of iodev>';'
```

Související informace

Pro informace o	Viz
Otevírání, atd. souborů	*Technická referenční příručka - Přehled RAPID*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.204 RMQEmptyQueue - Prázdná fronta zpráv RAPID

Použití

`RMQEmptyQueue` vyprazdňuje frontu zpráv RAPID (RMQ) v úloze, která vykonává instrukci.

Základní příklady

Následující příklad názorně ukazuje instrukci `RMQEmptyQueue`:

Příklad

```
RMQEmptyQueue;
```

Instrukce `RMQEmptyQueue` odstraňuje všechny zprávy z RMQ v prováděné úloze.

Vykonávání programu

Fronta zpráv RAPID vlastněná vykonávanou úlohou je vyprázdněna. Instrukci je možné používat na všech úrovních vykonávání.

Omezení

`RMQEmptyQueue` pouze vyprazdňuje frontu zpráv RAPID v úloze, která vykonává instrukci. Všechny ostatní fronty zpráv RAPID jsou ponechány beze změny.

Syntaxe

```
RMQEmptyQueue ';'
```

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
Datový typ `rmqmessage`	Kontrola informačních štítků
Odeslat data do fronty úlohy RAPID	Kontrola informačních štítků
Odeslat data do fronty úlohy RAPID a čekat na odpověď od klienta	Kontrola informačních štítků
Najít číslo identity úlohy fronty zpráv RAPID	Kontrola informačních štítků
Vyjímá hlavičku dat z `rmqmessage`	Kontrola informačních štítků
Vyjímá data z `rmqmessage`	Kontrola informačních štítků
Přikázat a zapnout přerušení pro určený datový typ	Kontrola informačních štítků
Získat jméno slotu od určené identity slotu	Kontrola informačních štítků
Přijmout zprávu od RMQ	Kontrola informačních štítků
Získat první zprávu z fronty zpráv RAPID	Kontrola informačních štítků

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.205 RMQFindSlot - Najít identitu slotu od jména slotu

Použití

`RMQFindSlot` (*RAPID Message Queue Find Slot*) se používá pro hledání identity slotu k RMQ konfigurovanému pro úlohu RAPID, nebo identity slotu ke klientu Robot Application Builder.

Základní příklady

Následující příklad názorně ukazuje instrukci `RMQFindSlot`:

Příklad 1

```
VAR rmqslot myrmqslot;
RMQFindSlot myrmqslot, "RMQ_T_ROB2";
```

Získat číslo identity pro RMQ`"RMQ_T_ROB2"` konfigurovaného pro úlohu RAPID `"T_ROB2"`.

Argumenty

```
RMQFindSlot Slot Name
```

`Slot`

Datový typ: `rmqslot`

Proměnná, do které je vrácen numerický identifikátor.

`Name`

Datový typ: `string`

Jméno klienta, pro kterého se bude hledat číslo identity. Jméno musí být správné, pokud se jedná o malá a velká písmena. Jestliže úloha RAPID je pojmenována T_ROB1 a použije se jméno RMQ_t_rob1 pro RMQ, skončí to chybou (viz kapitola o ošetřování chyb dole).

Vykonávání programu

Instrukce `RMQFindSlot` se používá pro nalezení identity slotu pro jmenovitý RMQ nebo klienta Robot Application Builder.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_RMQ_NAME`	Dané jméno slotu není platné nebo nebylo nalezeno.

Syntaxe

```
RMQFindSlot
  [ Slot ':=' ] < variable (VAR) of rmqslot > ','
  [ Name ':=' ] < expression (IN) of string > ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
Odeslat data do fronty úlohy RAPID	*RMQSendMessage - Odeslat datovou zprávu RMQ na str 568*
Získat první zprávu z fronty zpráv RAPID (RMQ).	*RMQGetMessage - Získat zprávu RMQ na str 556*
Odeslat data do fronty úlohy RAPID a čekat na odpověď od klienta	*RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu na str 572*
Vyjímá hlavičku dat z `rmqmessage`	*RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ na str 562*
Přikázat a zapnout přerušení pro určený datový typ	*IRMQMessage - Přikazuje přerušení RMQ pro datový typ na str 288*
Vyjímá data z `rmqmessage`	*RMQGetMsgData - Získat datovou část zprávy RMQ na str 559*
Získat jméno slotu od určené identity slotu	*RMQGetSlotName - Získat jméno klienta RMQ na str 1302*
`RMQ Slot`	*rmqslot - Číslo identity klienta RMQ na str 1570*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.206 RMQGetMessage - Získat zprávu RMQ

Použití

`RMQGetMessage` (*RAPID Message Queue Get Message*) se používá pro získání první RMQ zprávy z fronty pro aktuální programovou úlohu.

Základní příklady

Následující příklad názorně ukazuje instrukci `RMQGetMessage`:

Viz také *Další příklady na str 556*.

Příklad 1

```
TRAP msghandler
  VAR rmqmessage myrmqmsg;
  RMQGetMessage myrmqmsg;
  ...
ENDTRAP
```

V TRAP rutině `msghandler` se získá `rmqmessage` od RMQ a zkopíruje se do proměnné `myrmqmsg`.

Argumenty

```
RMQGetMessage Message
```

`Message`

Datový typ: `rmqmessage`

Proměnná pro uložení zprávy RMQ.

Maximální velikost dat, která lze přijmout v `rmqmessage`, je asi 3000 bajtů.

Vykonávání programu

Instrukce `RMQGetMessage` se používá k získání první zprávy z fronty úlohy, vykonávající instrukci. Jestliže je tam zpráva, bude zkopírována do proměnné `Message` a potom odstraněna z fronty, aby vznikl prostor pro nové zprávy. Instrukce je podporována pouze na úrovni TRAP.

Další příklady

Více příkladů jak používat instrukci `RMQGetMessage` je názorně uvedeno dole.

Příklad 1

```
RECORD mydatatype
  int x;
  int y;
ENDRECORD

VAR intnum msgreceive;
VAR mydatatype mydata;

PROC main()
  ! Setup interrupt
  CONNECT msgreceive WITH msghandler;
  ! Order cyclic interrupt to occur for data type mydatatype
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  IRMQMessage mydata, msgreceive;
  WHILE TRUE DO
    ! Performing cycle
    ...
  ENDWHILE
ENDPROC

TRAP msghandler
  VAR rmqmessage message;
  VAR rmqheader header;

  ! Get the RMQ message
  RMQGetMessage message;
  ! Copy RMQ header information
  RMQGetMsgHeader message \Header:=header;

  IF header.datatype = "mydatatype" AND header.ndim = 0 THEN
    ! Copy the data from the message
    RMQGetMsgData message, mydata;
  ELSE
    TPWrite "Received a type not handled or with wrong dimension";
  ENDIF
ENDTRAP
```

Když je přijata nová zpráva, je vykonána TRAP rutina `msghandler` a nová zpráva je zkopírována do proměnné `message` (instrukce `RMQGetMessage`). Potom jsou zkopírována data hlavičky RMQ (instrukce `RMQGetMsgHeader`). Jestliže zpráva je očekávaného datového typu a má správný rozměr, data jsou zkopírována do proměnné `mydata` (instrukce `RMQGetMsgData`).

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_RMQ_NOMSG`	Ve frontě není v této chvíli žádná zpráva. Jestliže se vykonává `RMQGetMessage` dvakrát v TRAP rutině, může toto nastat. Chyba může být rovněž generována, jestliže došlo k výpadku napájení mezi přikázáním TRAP a vykonáním instrukce `RMQGetMessage`. Zprávy v RMQ budou při výpadku napájení ztraceny.
`ERR_RMQ_INVMSG`	Tato chyba bude vyhozena, jestliže zpráva je neplatná. To se může stát například tehdy, jestliže aplikace PC odešle poškozenou zprávu.

Omezení

`RMQGetMessage` není podporováno na uživatelské úrovni vykonávání (tj. v servisních rutinách) nebo na normální úrovni vykonávání.

Maximální velikost dat, která lze přijmout v `rmqmessage`, je asi 3000 bajtů.

Doporučuje se znovu použít proměnnou datového typu `rmqmessage`, pokud je to možné, pro ušetření paměti RAPID.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.206 RMQGetMessage - Získat zprávu RMQ

FlexPendant Interface, PC Interface, or Multitasking

Pokračování

Syntaxe

```
RMQGetMessage
  [ Message ':=' ] < variable (VAR) of rmqmessage > ';'
```

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
Najít číslo identity úlohy fronty zpráv RAPID	*RMQFindSlot - Najít identitu slotu od jména slotu na str 554*
Odeslat data do fronty úlohy RAPID	*RMQSendMessage - Odeslat datovou zprávu RMQ na str 568*
Odeslat data do fronty úlohy RAPID a čekat na odpověď od klienta	*RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu na str 572*
Vyjímá hlavičku dat z `rmqmessage`	*RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ na str 562*
Vyjímá data z `rmqmessage`	*RMQGetMsgData - Získat datovou část zprávy RMQ na str 559*
Přikázat a zapnout přerušení pro určený datový typ	*IRMQMessage - Přikazuje přerušení RMQ pro datový typ na str 288*
Získat jméno slotu od určené identity slotu	*RMQGetSlotName - Získat jméno klienta RMQ na str 1302*
`RMQ Message`	*rmqmessage - Zpráva z fronty zpráv RAPID na str 1569*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.207 RMQGetMsgData - Získat datovou část zprávy RMQ

Použití

`RMQGetMsgData` (*RAPID Message Queue Get Message Data*) se používá pro získání aktuálních dat v rámci zprávy RMQ

Základní příklady

Následující příklad názorně ukazuje instrukci `RMQGetMsgData`:

Viz také *RMQGetMsgData - Získat datovou část zprávy RMQ na str 559*.

Příklad 1

```
VAR rmqmessage myrmqmsg;
VAR num data;
...
RMQGetMsgData myrmqmsg, data;
! Handle data
```

Data datového typu `num` jsou získána od proměnné `myrmqmsg` a uložena v proměnné `data`.

Argumenty

```
RMQGetMsgData Message Data
```

`Message`

Datový typ: `rmqmessage`

Proměnná obsahující přijatou zprávu RMQ.

`Data`

Datový typ: `anytype`

Proměnná očekávaného datového typu, použitá pro uložení přijatých dat.

Vykonávání programu

Instrukce `RMQGetMsgData` se používá pro získání aktuálních dat v rámci zprávy RMQ, jejich konverzi ze znakového formátu ASCII na binární data a zkompilování dat, aby bylo vidět, jestli je možné uložit je do proměnné určené v instrukci, a potom jejich zkopírování do proměnné.

Další příklady

Více příkladů jak používat instrukci `RMQGetMsgData` je názorně uvedeno dole.

Příklad 1

```
RECORD mydatatype
  int x;
  int y;
ENDRECORD

VAR intnum msgreceive;
VAR mydatatype mydata;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.207 RMQGetMsgData - Získat datovou část zprávy RMQ

FlexPendant Interface, PC Interface, or Multitasking

Pokračování

```
PROC main()
  ! Setup interrupt
  CONNECT msgreceive WITH msghandler;
  ! Order cyclic interrupt to occur for data type mydatatype
  IRMQMessage mydata, msgreceive;
  WHILE TRUE DO
    ! Performing cycle
    ...
  ENDWHILE
ENDPROC

TRAP msghandler
  VAR rmqmessage message;
  VAR rmqheader header;

  ! Get the RMQ message
  RMQGetMessage message;
  ! Copy RMQ header information
  RMQGetMsgHeader message \Header:=header;

  IF header.datatype = "mydatatype" AND header.ndim = 0 THEN
    ! Copy the data from the message
    RMQGetMsgData message, mydata;
  ELSE
    TPWrite "Received a type not handled or with wrong dimension";
  ENDIF
ENDTRAP
```

Když je přijata nová zpráva, je vykonána TRAP rutina `msghandler` a nová zpráva je zkopírována do proměnné `message` (instrukce `RMQGetMessage`). Potom jsou zkopírována data hlavičky RMQ (instrukce `RMQGetMsgHeader`). Jestliže zpráva je očekávaného datového typu a má správný rozměr, data jsou zkopírována do proměnné `mydata` (instrukce `RMQGetMsgData`).

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_RMQ_VALUE`	**Přijatá zpráva a datový typ použitý v argumentu `Data` nemají stejný datový typ.**
`ERR_RMQ_DIM`	**Datové typy jsou stejné, ale liší se rozměry mezi daty ve zprávě a proměnnou použitou v argumentu `Data`.**
`ERR_RMQ_MSGSIZE`	**Velikost přijatých dat je větší než max konfigurovaná velikost pro RMQ u přijímané úlohy.**
`ERR_RMQ_INVMSG`	**Tato chyba bude vyhozena, jestliže zpráva je neplatná. To se může stát například tehdy, jestliže aplikace PC odešle poškozenou zprávu.**

Syntaxe

```
RMQGetMsgData
  [ Message ':=' ] < variable (VAR) of rmqmessage > ','
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
[ Data ':=' ] < reference (VAR) of anytype > ';'
```

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
Najít číslo identity úlohy fronty zpráv RAPID	*RMQFindSlot - Najít identitu slotu od jména slotu na str 554*
Odeslat data do fronty úlohy RAPID	*RMQSendMessage - Odeslat datovou zprávu RMQ na str 568*
Získat první zprávu z fronty zpráv RAPID (RMQ).	*RMQGetMessage - Získat zprávu RMQ na str 556*
Odeslat data do fronty úlohy RAPID a čekat na odpověď od klienta	*RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu na str 572*
Vyjímá hlavičku dat z `rmqmessage`	*RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ na str 562*
Přikázat a zapnout přerušení pro určený datový typ	*IRMQMessage - Přikazuje přerušení RMQ pro datový typ na str 288*
Získat jméno slotu od určené identity slotu	*RMQGetSlotName - Získat jméno klienta RMQ na str 1302*
`RMQ Message`	*rmqmessage - Zpráva z fronty zpráv RAPID na str 1569*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.208 RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ

Použití

`RMQGetMsgHeader` (*RAPID Message Queue Get Message Header*) získává informace z hlavičky přijaté zprávy RMQ a ukládá je do proměnných typu `rmqheader`, `rmqslot` nebo `num`.

Základní příklady

Následující příklady názorně ukazují instrukci `RMQGetMsgHeader`:

Viz také *Další příklady na str 563*.

Příklad 1

```
VAR rmqmessage myrmqmsg;
VAR rmqheader myrmqheader;
...
RMQGetMsgHeader myrmqmsg, \Header:=myrmqheader;
```

V tomto příkladu je proměnná `myrmqheader` naplněna daty zkopírovanými z `rmqheader` části proměnné `myrmqmsg`.

Příklad 2

```
VAR rmqmessage rmqmessage1;
VAR rmqheader rmqheader1;
VAR rmqslot rmqslot1;
VAR num userdef := 0;
...
RRMQGetMsgHeader rmqmessage1 \Header:=rmqheader1 \SenderId:=rmqslot1
     \UserDef:=userdef;
```

V tomto příkladu jsou proměnné `rmqheader1`, `rmqslot1` a `userdef` naplněny daty zkopírovanými z proměnné `rmqmessage1`.

Argumenty

```
RMQGetMsgHeader Message [\Header] [\SenderId] [\UserDef]
```

`Message`

Datový typ: `rmqmessage`

Proměnná obsahující přijatou zprávu RMQ, ze které by měly být zkopírovány informace o zprávě.

`[\Header]`

Datový typ: `rmqheader`

Proměnná pro uložení informací hlavičky RMQ, které jsou zkopírovány z proměnné určené jako parametr `Message`.

`[\SenderId]`

Datový typ: `rmqslot`

Proměnná pro uložení informací o identitě odesílatele, které jsou zkopírovány z proměnné určené jako parametr `Message`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[\UserDef]`

User Defined data

Datový typ: `num`

Proměnná pro uložení uživatelsky definovaných dat, která jsou zkopírována z proměnné určené jako parametr `Message`**. Aby odesílatel mohl získat jakákoliv platná data v této proměnné, musí stanovit, že by to mělo být zahrnuto při odesílání zprávy RMQ . Jestliže to není použito, hodnota bude nastavena na -1.**

Vykonávání programu

Instrukce `RMQGetMsgHeader` **získává informace z hlavičky v rámci přijaté zprávy RMQ a kopíruje je do proměnných typu** `rmqheader`, `rmqslot` **nebo** `num` **podle použitých argumentů.**

Další příklady

Více příkladů jak používat instrukci `RMQGetMsgHeader` **je názorně uvedeno dole.**

Příklad 1

```
RECORD mydatatype
  int x;
  int y;
ENDRECORD

VAR intnum msgreceive;
VAR mydatatype mydata;

PROC main()
  ! Setup interrupt
  CONNECT msgreceive WITH msghandler;
  ! Order cyclic interrupt to occur for data type mydatatype
  IRMQMessage mydata, msgreceive;
  WHILE TRUE DO
    ! Performing cycle
    ...
  ENDWHILE
ENDPROC

TRAP msghandler
  VAR rmqmessage message;
  VAR rmqheader header;

  ! Get the RMQ message
  RMQGetMessage message;
  ! Copy RMQ header information
  RMQGetMsgHeader message \Header:=header;

  IF header.datatype = "mydatatype" AND header.ndim = 0 THEN
    ! Copy the data from the message
    RMQGetMsgData message, mydata;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.208 RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ

FlexPendant Interface, PC Interface, or Multitasking

Pokračování

```
    ELSE
      TPWrite "Received a type not handled or with wrong dimension";
    ENDIF
  ENDTRAP
```

Když je přijata nová zpráva, je vykonána TRAP rutina `msghandler` a nová zpráva je zkopírována do proměnné `message` (instrukce `RMQGetMessage`). Potom jsou zkopírována data hlavičky RMQ (instrukce `RMQGetMsgHeader`). Jestliže zpráva je očekávaného datového typu a má správný rozměr, data jsou zkopírována do proměnné `mydata` (instrukce `RMQGetMsgData`).

Syntaxe

```
RMQGetMsgHeader
  [ Message ':=' ] < variable (VAR) of rmqmessage > ','
  [ '\' Header ':=' < variable (VAR) of rmqheader >
  [ '\' SenderId ':=' < variable (VAR) of rmqslot >
  [ '\' UserDef ':=' < variable (VAR) of num > ';'
```

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
Najít číslo identity úlohy fronty zpráv RAPID	*RMQFindSlot - Najít identitu slotu od jména slotu na str 554*
Odeslat data do fronty úlohy RAPID	*RMQSendMessage - Odeslat datovou zprávu RMQ na str 568*
Získat první zprávu z fronty zpráv RAPID (RMQ).	*RMQGetMessage - Získat zprávu RMQ na str 556*
Odeslat data do fronty úlohy RAPID a čekat na odpověď od klienta	*RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu na str 572*
Vyjímá data z `rmqmessage`	*RMQGetMsgData - Získat datovou část zprávy RMQ na str 559*
Přikázat a zapnout přerušení pro určený datový typ	*IRMQMessage - Přikazuje přerušení RMQ pro datový typ na str 288*
Získat jméno slotu od určené identity slotu	*RMQGetSlotName - Získat jméno klienta RMQ na str 1302*
`RMQ Slot`	*rmqslot - Číslo identity klienta RMQ na str 1570*
`RMQ Header`	*rmqmessage - Zpráva z fronty zpráv RAPID na str 1569*
`RMQ Message`	*rmqheader - Hlavička fronty zpráv RAPID na str 1567*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.209 RMQReadWait - Vrací zprávu od RMQ

Použití

`RMQReadWait` se používá v synchronním režimu pro přijetí jakéhokoliv typu zprávy.

Základní příklady

Následující příklad názorně ukazuje instrukci `RMQReadWait`:

Viz také *Další příklady na str 565*.

Příklad

```
VAR rmqmessage myrmqmsg;
RMQReadWait myrmqmsg;
```

První zpráva ve frontě je přijata do proměnné `myrmqmsg`.

Argumenty

```
RMQReadWait Message [\TimeOut]
```

`Message`

Datový typ: `rmqmessage`

Proměnná, do které je umístěna přijatá zpráva.

`[\Timeout]`

Datový typ: `num`

Max množství času [s], kdy vykonávání programu čeká na zprávu. Jestliže tento čas vyprší před splněním podmínky, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_RMQ_TIMEOUT`. Jestliže zde není žádný chybový handler, vykonávání se zastaví. Je možné nastavit časování na 0 (nula) sekund, takže nedojde k žádnému čekání.

Jestliže se nepoužívá parametr `\Timeout`, doba čekání je 60 sek. Chcete-li čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Vykonávání programu

Všechny příchozí zprávy jsou umístěny do fronty a `RMQReadWait` zpracovává zprávy v pořadí FIFO (první dovnitř - první ven), vždy po jedné zprávě. Je na odpovědnosti uživatele, aby se fronta zcela nezaplnila a aby byl připraven zpracovat každý typ zprávy podporovaný Frontou zpráv RAPID.

Další příklady

Více příkladů jak používat instrukci `RMQReadWait` je názorně uvedeno dole.

Příklad 1

```
VAR rmqmessage myrmqmsg;
RMQReadWait myrmqmsg \TimeOut:=30;
```

První zpráva ve frontě je přijata do proměnné `myrmqmsg`. Jestliže během 30 sekund není přijata žádná zpráva, vykonávání programu je zastaveno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.209 RMQReadWait - Vrací zprávu od RMQ

RobotWare - OS

Pokračování

Příklad 2

```
PROC main()
  VAR rmqmessage myrmqmsg;
  FOR i FROM 1 TO 25 DO
    RMQReadWait myrmqmsg \TimeOut:=30;
    ...
  ENDFOR

ERROR
  IF ERRNO = ERR_RMQ_TIMEOUT THEN
    TPWrite "ERR_RMQ_TIMEOUT error reported";
    ...
  ENDIF
ENDPROC
```

Zprávy jsou přijímány z fronty a ukládány do proměnné `myrmqmsg`. Jestliže přijímání zprávy trvá déle než 30 sekund, je volán chybový handler.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

Kód chyby	Popis
`ERR_RMQ_TIMEOUT`	Ve stanoveném čase nebyla přijata žádná odpověď
`ERR_RMQ_INVMSG`	Tato chyba bude vyhozena, jestliže zpráva je neplatná. To se může stát například tehdy, jestliže aplikace PC odešle poškozenou zprávu

Omezení

`RMQReadWait` je podporováno pouze v synchro režimu. Vykonávání této instrukce v režimu založeném na přerušení způsobí fatální chybu za běhu.

`RMQReadWait` není podporováno na úrovni vykonávání trap nebo na uživatelské úrovni vykonávání. Vykonávání této instrukce na některé z těchto úrovní způsobí fatální chybu za běhu.

Syntaxe

```
RMQReadWait
  [ Message ':=' ] < variable (VAR) of rmqmessage>
  [ '\' TimeOut':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
Popis režimů vykonávání úloh	*Technická referenční příručka - Systémové parametry*
Datový typ `rmqmessage`	Kontrola informačních štítků
Odeslat data do fronty úlohy RAPID	Kontrola informačních štítků

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Odeslat data do fronty úlohy RAPID a čekat na odpověď od klienta	Kontrola informačních štítků
Najít číslo identity úlohy fronty zpráv RAPID	Kontrola informačních štítků
Vyjímá hlavičku dat z `rmqmessage`	Kontrola informačních štítků
Vyjímá data z `rmqmessage`	Kontrola informačních štítků
Přikázat a zapnout přerušení pro určený datový typ	Kontrola informačních štítků
Získat jméno slotu od určené identity slotu	Kontrola informačních štítků
Prázdná fronta zpráv RAPID	Kontrola informačních štítků
Získat první zprávu z fronty zpráv RAPID	Kontrola informačních štítků

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.210 RMQSendMessage - Odeslat datovou zprávu RMQ

Použití

`RMQSendMessage` (*RAPID Message Queue Send Message*) se používá pro odeslání dat k RMQ konfigurovanému pro úlohu RAPID, nebo ke klientu Robot Application Builder.

Základní příklady

Následující příklady názorně ukazují instrukci `RMQSendMessage`:

Viz také *Další příklady na str 569*.

Příklad 1

```
VAR rmqslot destination_slot;
VAR string data:="Hello world";
..
RMQFindSlot destination_slot,"RMQ_Task2";
RMQSendMessage destination_slot,data;
```

Příklad ukazuje, jak odeslat hodnotu v proměnné `data` k úloze RAPID `"Task2"` s konfigurovaným RMQ `"RMQ_Task2"`.

Příklad 2

```
VAR rmqslot destination_slot;
CONST robtarget p5:=[ [600, 500, 225.3], [1, 0, 0, 0], [1, 1, 0,
     0], [ 11, 12.3, 9E9, 9E9, 9E9, 9E9] ];
VAR num my_id:=1;
..
RMQFindSlot destination_slot,"RMQ_Task2";
RMQSendMessage destination_slot, p5 \UserDef:=my_id;
my_id:=my_id + 1;
```

Příklad ukazuje, jak odeslat hodnotu v konstantě `p5` k úloze RAPID `"Task2"` s konfigurovaným RMQ "RMQ_Task2". Odesláno je také uživatelsky definované číslo. Toto číslo může použít příjemce jako identifikátor.

Argumenty

```
RMQSendMessage Slot SendData [\UserDef]
```

`Slot`

Datový typ: `rmqslot`

Číslo slotu identity klienta, který by měl přijmout zprávu.

`SendData`

Datový typ: `anytype`

Reference k proměnné, perzistentu nebo konstantě obsahující data, která budou odeslána ke klientu s identitou jako v argumentu `Slot`.

`[\UserDef]`

User Defined data

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Data určující uživatelsky definované informace k příjemci `SendData`, tj. klientu s číslem identity jako v proměnné `Slot`. Hodnota musí být celé číslo mezi 0 a 32767.

Vykonávání programu

Instrukce `RMQSendMessage` se používá k odeslání dat určenému klientu. Instrukce má zabalená indata v ukládacím kontejneru a odesílá je.

Jestliže přijímací klient nemá zájem o příjem zpráv, tj. nemá nastavené žádné přerušení, které by nastalo pro datový typ určený v instrukci `RMQSendMessage` nebo nečeká v instrukci `RMQSendWait`, zpráva bude zrušena a bude vydáno varování.

Všechny datové typy není možné odesílat s instrukcí (viz omezení).

Další příklady

Více příkladů jak používat instrukci `RMQSendMessage` je názorně uvedeno dole.

Příklad 1

```
MODULE SenderMod
  RECORD msgrec
    num x;
    num y;
  ENDRECORD

  PROC main()
    VAR rmqslot destinationSlot;
    VAR msgrec msg :=[0, 0, 0];

    ! Connect to a Robot Application Builder client
    RMQFindSlot destinationSlot ”My_RAB_client”;

    ! Perform cycle
    WHILE TRUE DO
      ! Update msg with valid data
      ...
      ! Send message
      RMQSendMessage destinationSlot, msg;
      ...
    ENDWHILE
  ERROR
    IF ERRNO = ERR_RMQ_INVALID THEN
      ! Handle destination client lost
      WaitTime 1;
      ! Reconnect to Robot Application Builder client
      RMQFindSlot destinationSlot ”My_RAB_client”;
      ! Avoid execution stop due to retry count exceed
      ResetRetryCount;
      RETRY;
    ELSIF ERRNO = ERR_RMQ_FULL THEN
      ! Handle destination queue full
      WaitTime 1;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.210 RMQSendMessage - Odeslat datovou zprávu RMQ

FlexPendant Interface, PC Interface, or Multitasking

Pokračování

```
        ! Avoid execution stop due to retry count exceed
        ResetRetryCount;
        RETRY;
      ENDIF
    ENDPROC
ENDMODULE
```

Příklad ukazuje způsob používání instrukce `RMQSendMessage` s ošetřováním vzniklých chyb za běhu. Program odesílá uživatelsky definovaná data typu `msgrec` ke klientu Robot Application Builder nazvanému "`My_RAB_client`".

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_RMQ_MSGSIZE`	Velikost zprávy je příliš velká. Data buď překračují max přípustnou velikost zprávy nebo přijímací klient není konfigurován pro příjem takové velikosti dat, jaká je odesílána.
`ERR_RMQ_FULL`	Cílová fronta zpráv je plná
`ERR_RMQ_INVALID`	Cílový slot nebyl připojen nebo cílový slot už není dostupný. Jestliže není připojen, musí být provedeno volání k `RMQFindSlot`. Jestliže není dostupný, důvodem je odpojení vzdáleného klienta od řadiče.

Omezení

Není možné nastavovat přerušení, odesílat nebo přijímat datové instance datových typů, které jsou nehodnotové, polohodnotové typy nebo datové typy `motsetdata`

Max velikost dat, která mohou být odeslána ke klientu Robot Application Builder je asi 5000 bajtů. Max velikost dat, která mohou být přijata RMQ a uložena v datovém typu `rmqmessage` je asi 3000 bajtů. Velikost dat, která mohou být přijata RMQ, může být konfigurována (výchozí velikost 400, max velikost 3000).

Syntaxe

```
RMQSendMessage
  [ Slot ':=' ] < variable (VAR) of rmqslot > ','
  [ SendData ':=' ] < reference (REF) of anytype >
  [ '\' UserDef ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
Najít číslo identity úlohy fronty zpráv RAPID	*RMQFindSlot - Najít identitu slotu od jména slotu na str 554*
Získat první zprávu z fronty zpráv RAPID (RMQ).	*RMQGetMessage - Získat zprávu RMQ na str 556*
Odeslat data do fronty úlohy RAPID a čekat na odpověď od klienta	*RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu na str 572*
Vyjímá hlavičku dat z `rmqmessage`	*RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ na str 562*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Vyjímá data z `rmqmessage`	*RMQGetMsgData - Získat datovou část zprávy RMQ na str 559*
Přikázat a zapnout přerušení pro určený datový typ	*IRMQMessage - Přikazuje přerušení RMQ pro datový typ na str 288*
Získat jméno slotu od určené identity slotu	*RMQGetSlotName - Získat jméno klienta RMQ na str 1302*
`RMQ Slot`	*rmqslot - Číslo identity klienta RMQ na str 1570*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.211 RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu

Použití

S instrukcí `RMQSendWait` (*RAPID Message Queue Send Wait*) je možné odesílat data k RMQ nebo klientu Robot Application Builder a čekat na odpověď od určeného klienta. Při používání této instrukce potřebuje uživatel vědět, jaký typ dat bude odeslán v odpovědi od klienta.

Základní příklady

Následující příklady názorně ukazují instrukci `RMQSendWait`:

Viz také *Další příklady na str 574*.

Příklad 1

```
VAR rmqslot destination_slot;
VAR string sendstr:="This string is from T_ROB1";
VAR rmqmessage receivemsg;
VAR num mynum;
..
RMQFindSlot destination_slot, "RMQ_T_ROB2";
RMQSendWait destination_slot, sendstr, receivemsg, mynum;
RMQGetMsgData receivemsg, mynum;
```

Příklad ukazuje, jak odesílat data v proměnné `sendstr` k úloze RAPID "`T_ROB2`" s konfigurovaným RMQ "`RMQ_T_ROB2`". Nyní instrukce `RMQSendWait` čeká na odpověď od úlohy "T_ROB2". Instrukce v "T_ROB2“ potřebuje odeslat data, která jsou uložena v datovém typu `num` pro ukončení instrukce čekání `RMQSendWait`. Když je zpráva přijata, data jsou zkopírována do proměnné `mynum` z proměnné `receivemsg` s instrukcí `RMQGetMsgData`.

Příklad 2

```
VAR rmqslot rmqslot1;
VAR string mysendstr;
VAR rmqmessage rmqmessage1;
VAR string receivestr;
VAR num mysendid:=1;
..
mysendstr:="Message from Task1";
RMQFindSlot rmqslot1, "RMQ_Task2";
RMQSendWait rmqslot1, mysendstr \UserDef:=mysendid, rmqmessage1,
     receivestr \TimeOut:=20;
RMQGetMsgData rmqmessage1, receivestr;
mysendid:=mysendid + 1;
```

Příklad ukazuje, jak odesílat data v proměnné `mysendstr` k úloze RAPID "`Task2`" s konfigurovaným RMQ"`RMQ_Task2`". Uživatelsky definované číslo je také odesláno. Toto číslo může použít příjemce jako identifikátor a musí být vráceno zpět odesílateli kvůli ukončení čekání instrukce `RMQSendWait`. Dalším požadavkem k ukončení čekací instrukce je, aby správný datový typ byl odeslán od klienta. Tento datový typ je určen proměnnou `receivestr` v instrukci `RMQSendWait`. Po

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

přijetí zprávy jsou aktuální data zkopírována do proměnné `receivestr` s instrukcí `RMQGetMsgData`.

Argumenty

```
RMQSendWait Slot SendData [\UserDef] Message ReceiveDataType
     [\TimeOut]
```

`Slot`

Datový typ: `rmqslot`

Číslo identity klienta, který by měl přijmout zprávu.

`SendData`

Datový typ: `anytype`

Reference k proměnné, perzistentu nebo konstantě obsahující data, která budou odeslána ke klientu s číslem identity jako v proměnné `Slot`.

`[\UserDef]`

User Defined data

Datový typ: `num`

Data určující uživatelsky definované informace k příjemci `SendData`, to znamená klientu s číslem identity jako v proměnné `Slot`. Při použití tohoto volitelného argumentu bude instrukce `RMQSendWait` pouze ukončovat, jestliže `ReceiveDataType` a určený `UserDef` je jak bylo určeno v odpovědi na zprávu. Hodnota musí být celé číslo mezi 0 a 32767.

`Message`

Datový typ: `rmqmessage`

Proměnná, do které je umístěna přijatá zpráva.

`ReceiveDataType`

Datový typ: `anytype`

Reference k perzistentu, proměnné nebo konstantě datového typu, na který čeká instrukce. Aktuální data nejsou zkopírována do této proměnné, když je vykonáván `RMQSendWait`. Tento argument je použit pouze pro určení aktuálního datového typu, na který čeká instrukce `RMQSendWait`.

`[\Timeout]`

Datový typ: `num`

Max množství času [s], kdy vykonávání programu čeká na odpověď. Jestliže tento čas vyprší před splněním podmínky, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_RMQ_TIMEOUT`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

Jestliže se nepoužívá parametr `\Timeout`, doba čekání je 60 sek. Chcete-li čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.211 RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu

FlexPendant Interface, PC Interface, or Multitasking
Pokračování

Vykonávání programu

Instrukce `RMQSendWait` odesílá data a čeká na odpověď od klienta s určenou identitou slotu. Odpověď musí být `rmqmessage` od klienta, který dostal zprávu a odpověď musí být stejného datového typu, který je určen v argumentu `ReceiveDataType`. Odpověď bude odeslána stejným způsobem jako při použití `RMQSendMessage`, tzn. příjemce dostane normální zprávu fronty zpráv RAPID. Je na odpovědnosti odesílatele, aby příjemce věděl, že se požaduje odpověď. Jestliže se použije volitelný argument `UserDef` v `RMQSendWait`, požadavkem je, aby přijímací klient použil stejný `UserDef` v odpovědi.

Jestliže přijímací klient nemá zájem na přijímání zpráv, tzn. nemá nastavené žádné přerušení pro datový typ určený v instrukci `RMQSendWait`, zpráva bude zrušena a bude vydáno varování. Instrukce vrací chybu po době použité v argumentu `TimeOut`, nebo po výchozím čase vypršení 60 s. Tato chyba může být ošetřena v chybovém handleru.

Instrukce `RMQSendWait` má nejvyšší prioritu, jestliže zpráva je přijata a souhlasí s popisem očekávané odpovědi a zprávy připojené k TRAP rutině (viz instrukce `IRMQMessage`.

Jestliže dojde k výpadku napájení při čekání na odpověď od klienta, proměnná použitá v argumentu `Slot` se nastaví na 0 a instrukce je vykonána znovu. Instrukce potom selže kvůli neplatné identitě slotu a bude volán chybový handler, pokud existuje, s kódem chyby `ERR_RMQ_INVALID`. Identita slotu tam může být znovu inicializována.

Všechny datové typy není možné odesílat s instrukcí (viz omezení).

Další příklady

Více příkladů jak používat instrukci `RMQSendWait` je názorně uvedeno dole.

Příklad 1

```
MODULE RMQ_Task1_mod
  PROC main()
    VAR rmqslot destination_slot;
    VAR string mysendstr:="String sent from RMQ_Task1_mod";
    VAR string myrecstr;
    VAR rmqmessage recmsg;
    VAR rmqheader header;

    !Get slot identity to client called RMQ_Task2
    RMQFindSlot destination_slot, "RMQ_Task2";

    WHILE TRUE DO
      ! Do something
      ...
      !Send data in mysendstr, wait for an answer of type string
      RMQSendWait destination_slot, mysendstr, recmsg, myrecstr;
      !Get information about the received message
      RMQGetMsgHeader recmsg \Header:=header;
      IF header.datatype = "string" AND header.ndim = 0 THEN
        ! Copy the data in recmsg
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
      RMQGetMsgData recmsg, myrecstr;
      TPWrite "Received string: " + myrecstr;
    ELSE
      TPWrite "Not a string that was received";
    ENDIF
  ENDWHILE
ENDPROC
ENDMODULE
```

Data v proměnné `mysendstr` jsou odeslána k úloze RAPID "`Task2`" s konfigurovanou frontou zpráv RAPID "`RMQ_Task2`" s instrukcí `RMQSendWait`. Odpověď od úlohy RAPID "`Task2`" by měl být řetězec (určený datovým typem proměnné `myrecstr`). Zpráva RMQ přijatá jako odpověď je přijata v proměnné `recmsg`. Použití proměnné `myrecstr` ve volání k `RMQSendWait` je pouze specifikací datového typu, který příjemce očekává jako odpověď. Žádná platná data nejsou umístěna do proměnné ve volání `RMQSendWait`.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_RMQ_MSGSIZE`	Velikost zprávy je příliš velká. Data buď překračují max přípustnou velikost zprávy nebo přijímací klient není konfigurován pro příjem takové velikosti dat, jaká je odesílána.
`ERR_RMQ_FULL`	Cílová fronta zpráv je plná.
`ERR_RMQ_INVALID`	`rmqslot` nebyl inicializován nebo cílový slot už není dostupný. To se může stát, když cílový slot je vzdáleným klientem a vzdálený klient se odpojil od řadiče. `RMQSendWait` byl přerušen výpadkem napájení a při restartu je `rmqslot` nastaven na 0.
`ERR_RMQ_TIMEOUT`	Ve stanoveném čase nebyla přijata žádná odpověď.
`ERR_RMQ_INVMSG`	Tato chyba bude vyhozena, jestliže zpráva je neplatná. To se může stát například tehdy, jestliže aplikace PC odešle poškozenou zprávu.

Omezení

Není dovoleno vykonávat `RMQSendWait` v synchronizovaném režimu. Způsobí to fatální chybu při běhu.

Není možné nastavovat přerušení, odesílat nebo přijímat datové instance datových typů, které jsou nehodnotové, polohodnotové typy nebo datové typy `motsetdata`

Max velikost dat, která mohou být odeslána ke klientu Robot Application Builder je asi 5000 bajtů. Max velikost dat, která mohou být přijata RMQ a uložena v datovém typu `rmqmessage` je asi 3000 bajtů. Velikost dat, která mohou být přijata RMQ, může být konfigurována (výchozí velikost 400, max velikost 3000).

Syntaxe

```
RMQSendWait
  [ Slot ':=' ] < variable (VAR) of rmqslot > ','
  [ SendData ':=' ] < reference (REF) of anytype >
  [ '\' UserDef ':=' < expression (IN) of num > ] ','
  [ Message' :=' ] < variable (VAR) of rmqmessage > ','
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.211 RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu

FlexPendant Interface, PC Interface, or Multitasking

Pokračování

```
[ ReceiveDataType ':=' ] < reference (REF) of anytype > ','
[ '\' Timeout ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
Najít číslo identity úlohy fronty zpráv RAPID	*RMQFindSlot - Najít identitu slotu od jména slotu na str 554*
Odeslat data do fronty úlohy RAPID	*RMQSendMessage - Odeslat datovou zprávu RMQ na str 568*
Získat první zprávu z fronty zpráv RAPID (RMQ).	*RMQGetMessage - Získat zprávu RMQ na str 556*
Vyjímá hlavičku dat z `rmqmessage`	*RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ na str 562*
Vyjímá data z `rmqmessage`	*RMQGetMsgData - Získat datovou část zprávy RMQ na str 559*
Přikázat a zapnout přerušení pro určený datový typ	*IRMQMessage - Přikazuje přerušení RMQ pro datový typ na str 288*
Získat jméno slotu od určené identity slotu	*RMQGetSlotName - Získat jméno klienta RMQ na str 1302*
`RMQ Slot`	*rmqslot - Číslo identity klienta RMQ na str 1570*
`RMQ Message`	*rmqmessage - Zpráva z fronty zpráv RAPID na str 1569*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.212 SafetyControllerSyncRequest - Iniciace hardwarové synchronizační procedury

Použití

`SafetyControllerSyncRequest` se používá pro iniciaci hardwarové synchronizační procedury.

Základní příklady

Následující příklad názorně ukazuje instrukci `SafetyControllerSyncRequest`.

Příklad 1

```
SafetyControllerSyncRequest;
```

Iniciovat hardwarovou synchronizační proceduru.

Vykonávání programu

Tato instrukce musí být volána před aktivací synchronizačního signálu.

Syntaxe

```
SafetyControllerSyncRequest ';'
```

Související informace

Pro informace o	Viz
`SafetyControllerGetChecksum`	*SafetyControllerGetChecksum - Získat kontrolní součet pro uživatelsky konfigurovaný soubor na str 1313*
`SafetyControllerGetSWVersion`	*SafetyControllerGetSWVersion - Získat verzi firmwaru bezpečnostního řadiče na str 1314*
`SafetyControllerGetUserChecksum`	*SafetyControllerGetUserChecksum - Získat kontrolní součet pro chráněné parametry na str 1315*
Bezpečnostní konfiguraceSafeMove	*Application manual - Functional safety and SafeMove*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.213 Save - Uložit programový modul

Použití

`Save` se používá pro uložení programového modulu.

Určený programový modul v paměti programu bude uložen s původní (určenou v `Load` or `StartLoad`) nebo určenou cestou souboru.

Je také možné uložit systémový modul na určenou cestu souboru.

Základní příklady

Následující příklad názorně ukazuje instrukci `Save`:

Viz také *Další příklady na str 579*.

Příklad 1

```
Load "HOME:/PART_B.MOD";
...
Save "PART_B";
```

Načíst programový modul se jménem souboru `PART_B.MOD` z `HOME:` do paměti programu.

Uložit programový modul `PART_B` s původní cestou souboru `HOME:` a s původním jménem souboru `PART_B.MOD`.

Argumenty

```
Save [\TaskRef]|[\TaskName] ModuleName [\FilePath] [\File]
```

`[\TaskRef]`

Task Reference

Datový typ: `taskid`

Identita programové úlohy, do které by měl být programový modul uložen.

Pro všechny programové úlohy v systému budou dostupné předdefinované proměnné datového typu `taskid`. Identita proměnné bude například "taskname"+"Id", pro úlohu `T_ROB1` bude identita proměnné `T_ROB1Id`.

`[\TaskName]`

Datový typ: `string`

Jméno programové úlohy, do které by měl být programový modul uložen.

Jestliže není určen žádný z argumentů `\TaskRef` nebo `\TaskName`, potom bude uložen určený programový modul v aktuální (vykonávané) programové úloze.

`ModuleName`

Datový typ: `string`

Programový modul k uložení.

`[\FilePath]`

Datový typ: `string`

Cesta souboru a jméno souboru k místu, kde bude uložen programový modul. Jméno souboru by mělo být vyřazeno, když se používá argument `\File`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[\File]`

Datový typ: `string`

Když je jméno souboru vyřazeno v argumentu `\FilePath`, potom musí být určeno s tímto argumentem.

Argument `\FilePath\File` může být vynechán pouze u programových modulů načtených s `Load` nebo `StartLoad-WaitLoad` a programový modul bude uložen do stejného místa, jako které je určeno v těchto instrukcích. Chcete-li uložit programový modul do jiného místa, je možné použít argument `\FilePath \File`.

Argument `\FilePath \File` se musí použít, aby bylo možné uložit programový modul, který byl dříve načten z FlexPendantu, externího počítače nebo systémové konfigurace.

Vykonávání programu

Vykonávání programu čeká, až programový modul dokončí ukládání, potom pokračuje s další instrukcí.

Další příklady

Více příkladů jak používat instrukci `Save` je názorně uvedeno dole.

Příklad 1

```
Save "PART_A" \FilePath:="HOME:/DOORDIR/PART_A.MOD";
```

Uložit programový modul `PART_A` do `HOME:` v souboru `PART_A.MOD` a v adresáři `DOORDIR`.

Příklad 2

```
Save "PART_A" \FilePath:="HOME:" \File:="DOORDIR/PART_A.MOD";
```

Stejné jako v příkladu 1 nahoře, ale jiná syntaxe.

Příklad 3

```
Save \TaskRef:=TSK1Id, "PART_A"
     \FilePath:="HOME:/DOORDIR/PART_A.MOD";
```

Uložit programový modul `PART_A` v programové úloze `TSK1` do určené destinace. Toto je příklad, kde se vykonává instrukce `Save` v jedné programové úloze a ukládání je provedeno do jiné programové úlohy.

Příklad 4

```
Save \TaskName:="TSK1", "PART_A"
     \FilePath:="HOME:/DOORDIR/PART_A.MOD";
```

Uložit programový modul `PART_A` v programové úloze `TSK1` do určené destinace. Toto je další příklad, kde se vykonává instrukce `Save` v jedné programové úloze a ukládání je provedeno do jiné programové úlohy.

Omezení

TRAP rutiny, systémové I/O události, a jiné programové úlohy nemohou být vykonávány během operace ukládání. Proto bude každá taková operace odložena.

Operace ukládání může přerušit aktualizaci dat PERS prováděných krok za krokem z jiných programových úloh. Výsledkem budou nekonzistentní data PERS .

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.213 Save - Uložit programový modul

RobotWare - OS

Pokračování

Zastavení programu během instrukce Save má za výsledek zastavení ochrany s vypnutím motorů a chybovou zprávou "20025 Stop order timeout" na FlexPendantu.

Vyloučit probíhající pohyby robotu během ukládání.

Řešení chyb

Jestliže jméno programové úlohy v argumentu `\TaskName` není možné nalézt v systému, systémová proměnná `ERRNO` se nastaví na `ERR_TASKNAME`.

Jestliže není možné uložit programový modul, protože neexistuje jméno modulu, neznámé nebo dvojznačné jméno modulu, potom je systémová proměnná `ERRNO` nastavena na `ERR_MODULE`.

Jestliže není možné otevřít uložený soubor, protože bylo odepřeno oprávnění, adresář neexistuje nebo nezbylo žádné místo na zařízení, potom je systémová proměnná `ERRNO` nastavena na `ERR_IOERROR`.

Jestliže argument `\FilePath` není určen pro programové moduly načtené z FlexPendantu, systémových parametrů nebo externího počítače, potom je systémová proměnná `ERRNO` nastavena na `ERR_PATH`.

Chyby uvedené shora mohou být ošetřeny v chybovém handleru.

Syntaxe

```
Save
  [[ ’\’ TaskRef ’:=’ <variable (VAR) of taskid>]
  |[ ’\’ TaskName’ :=’ <expression (IN) of string>] ’,’]
  [ ModuleName’ :=’ ] <expression (IN) of string>
  [ ’\’ FilePath’ :=’<expression (IN) of string> ]
  [ ’\’ File’ :=’ <expression (IN) of string>] ’;’
```

Související informace

Pro informace o	Viz
Programové úlohy	*taskid - Identifikace úlohy na str 1606*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.214 SaveCfgData - Uložit systémové parametry do souboru

Použití

`SaveCfgData` se používá pro uložení systémových parametrů do souboru. Může to být výhodné po aktualizaci systémových parametrů s instrukcí `WriteCfgData`.

Základní příklady

Následující příklady názorně ukazují instrukci `SaveCfgData`.

Příklad 1

```
SaveCfgData "SYSPAR" \File:="MYEIO.cfg", EIO_DOMAIN;
```

Ukládání I/O konfigurační domény do souboru `MYEIO.cfg` v adresáři `SYSPAR`.

Příklad 2

```
SaveCfgData "SYSPAR", ALL_DOMAINS;
```

Ukládání všech existujících konfiguračních domén do adresáře `SYSPAR`. Soubory dostanou jména `EIO.cfg`, `MMC.cfg`, `PROC.cfg`, `SIO.cfg`, `SYS.cfg` a `MOC.cfg`.

Argumenty

```
SaveCfgData FilePath [\File] Domain
```

`FilePath`

Datový typ: `string`

Cesta souboru a jméno souboru k místu, kde bude soubor uložen. Jméno souboru by mělo být vyřazeno, když se používá argument `\File`.

`[\File]`

Datový typ: `string`

Když je jméno souboru vyřazeno v argumentu `\FilePath`, potom musí být určeno s tímto argumentem.

`Domain`

Datový typ: `cfgdomain`

Doména systémového parametru k uložení.

Vykonávání programu

Ukládá systémové parametry do souboru.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_CFG_ILL_DOMAIN`	Použitá `cfgdomain` je neplatná nebo se nepoužívá.
`ERR_CFG_WRITEFILE`	Adresář neexistuje nebo `FilePath` a `File` použité v adresáři, nebo nějaký jiný problém v souvislosti s ukládáním souboru.

Syntaxe

```
SaveCfgData
  [FilePath ':=' ] <expression (IN) of string>
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.214 SaveCfgData - Uložit systémové parametry do souboru

Pokračování

```
['\' File ':=' <expression (IN) of string>]
[Domain ':=' ] <expression (IN) of cfgdomain> ';'
```

Související informace

Pro informace o	Viz
Data`cfgdomain`	*cfgdomain - Konfigurační doména na str 1471*
Systémové parametry	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.215 SCWrite - Odeslat variabilní data k aplikaci klienta

Použití

`SCWrite` (*Superior Computer Write*) se používá k odeslání jména, typu, dimenze a hodnoty perzistentní proměnné k aplikaci klienta. Je možné odeslat jak jednoduché proměnné, tak i pole proměnných.

Základní příklady

Následující příklady názorně ukazují instrukci `SCWrite`:

Příklad 1

```
PERS num cycle_done;

PERS num numarr{2}:=[1,2];

SCWrite cycle_done;
```

Jméno, typ a hodnota perzistentní proměnné `cycle_done` jsou odeslány ke všem aplikacím klienta.

Příklad 2

```
SCWrite \ToNode := "138.221.228.4", cycle_done;
```

Jméno, typ a hodnota perzistentní proměnné `cycle_done` jsou odeslány ke všem aplikacím klienta. Argument `\ToNode` bude ignorován.

Příklad 3

```
SCWrite numarr;
```

Jméno, typ, dim a hodnota perzistentní proměnné `numarr` jsou odeslány ke všem aplikacím klienta.

Příklad 4

```
SCWrite \ToNode := "138.221.228.4", numarr;
```

Jméno, typ, dim a hodnota perzistentní proměnné `numarr` jsou odeslány ke všem aplikacím klienta. Argument `\ToNode` bude ignorován.

Argumenty

```
SCWrite [ \ToNode ] Variable
```

`[\ToNode]`

Datový typ: `datatype`

Argument bude ignorován.

`Variable`

Datový typ: `anytype`

Jméno perzistentní proměnné.

Vykonávání programu

Jméno, typ, dim a hodnota perzistentní proměnné jsou odeslány ke všem aplikacím klienta. ‘dim’ je dimenze proměnné a je odeslána pouze v případě, že proměnná je polem.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.215 SCWrite - Odeslat variabilní data k aplikaci klienta
PC interface/backup
Pokračování

Řešení chyb

Instrukce `SCWrite` vrátí chybu v následujících případech:

Není možné odeslat proměnnou ke klientu. Může to míst následující příčiny:

- Zprávy `SCWrite` jsou tak stěsnané, že nemohou být odeslány ke klientu. Řešení: Vložte instrukci `WaitTime` mezi instrukce `SCWrite`.
- Hodnota proměnné je příliš velká a zmenšuje velikost `ARRAY` nebo `RECORD`.
- Chybová zpráva bude: `41473 System access error, Failed to send variable arg1`, kde `arg1` je jméno proměnné.

Když se objeví chyba, program se zastaví a musí být restartován. Systémová proměnná `ERRNO` bude obsahovat hodnotu `ERR_SC_WRITE`.

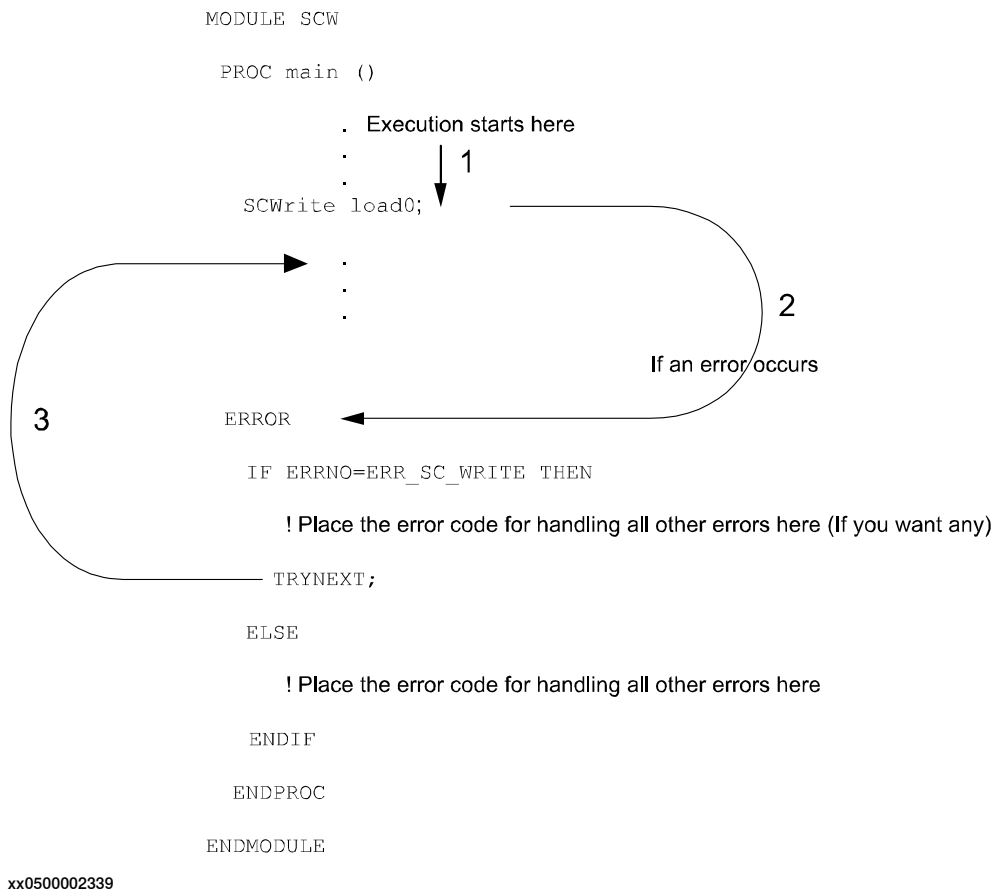
Instrukce `SCWrite` nevrátí chybu, jestliže aplikace klienta může být např. zavřena nebo komunikace neběží. Program bude pokračovat ve vykonávání.

Obnovení po chybě `SCWrite`

Aby se program nezastavil po vzniku chyby v instrukci `SCWrite`, musí být provedeno ošetření *chybovým handlerem*. Chyba bude pouze nahlášena do protokolu a program bude pokračovat v běhu.

Všimněte si, že ošetřování chyb ztíží hledání chyb v komunikaci klienta, protože chyba není nikdy hlášena na displej FlexPendantu (ale může být nalezena v protokolu).

Program RAPID vypadá následovně:



xx0500002339

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.216 SearchC - Hledá kruhově pomocí robotu

Použití

`SearchC` (*Search Circular*) se používá pro vyhledání pozice při kruhovém pohybu středním bodem nástroje (TCP).

Během pohybu robot dohlíží na digitální vstupní signál nebo perzistentní proměnnou. Když se hodnota signálu nebo perzistentní proměnné mění na požadovanou hodnotu, robot okamžitě přečte aktuální pozici.

Tuto instrukci je možné používat typicky, když nástroj držený robotem je sonda pro detekci povrchu. Vnější souřadnice pracovního objektu je možné získat pomocí instrukce `SearchC`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Při používání vyhledávacích instrukcí je důležité konfigurovat I/O systém, abychom měli velmi krátký čas od nastavení fyzického signálu k systému a dostali tak informace o nastavení (použijte jednotku I/O s kontrolou přerušení, nikoliv kontrolu volby). Způsob, jak to udělat, se může mezi aplikačními sběrnicemi lišit. Při používání DeviceNet potom ABB jednotky DSQC 651 (AD Combi I/O) a DSQC 652 (Digital I/O) budou dávat krátké časy, jelikož používají spojovací typ Změna stavu. Při použití jiných aplikačních sběrnic zajistěte konfiguraci sítě řádným způsobem, aby byly získány správné podmínky.

Základní příklady

Následující příklady názorně ukazují instrukci `SearchC`:

Viz také *Další příklady na str 591*.

Příklad 1

```
SearchC di1, sp, cirpoint, p10, v100, probe;
```

TCP `probe` je v pohybu kruhově k pozici `p10` rychlostí `v100`. Když se hodnota signálu `di1` změní na aktivní, pozice se uloží do `sp`.

Příklad 2

```
SearchC \Stop, di2, sp, cirpoint, p10, v100, probe;
```

TCP `probe` je v pohybu kruhově k pozici `p10`. Když se hodnota signálu `di2` změní na aktivní, pozice se uloží do `sp` a robot se okamžitě zastaví.

Příklad 3

```
PERS bool mypers:=FALSE;
...
SearchC \Stop, mypers, sp, cirpoint, p10, v100, probe;
```

TCP `probe` je v pohybu kruhově k pozici `p10`. Když se hodnota perzistentní proměnné `mypers`změní na TRUE, pozice se uloží do `sp` a robot se okamžitě zastaví.

Argumenty

```
SearchC [\Stop] | [\PStop] | [\SStop] | [\Sup] Signal | PersBool
     [\Flanks] | [\PosFlank] | [\NegFlank] | [\HighLevel] |
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.216 SearchC - Hledá kruhově pomocí robotu

RobotWare - OS

Pokračování

```
[\LowLevel] SearchPoint CirPoint ToPoint [\ID] Speed [\V] |
[\T] Tool [\WObj] [\Corr] [\TLoad]
```

`[ \Stop ]`

Stiff Stop

Datový typ: `switch`

Pohyb robotu je zastaven tak rychle, jak je to možné, bez podržení TCP na dráze (tvrdé zastavení), když se hodnota vyhledávacího signálu změní na aktivní nebo když se hodnota perzistentní proměnné změní na TRUE. Robot je posunut na malou vzdálenost, potom se zastaví a není posunut zpět k hledané pozici, tj. k pozici, kde se signál nebo perzistentní proměnná změnily.

VAROVÁNÍ

Zastavení hledání tuhým zastavením (přepínač `\Stop`) je dovoleno pouze v případě, že rychlost TCP je nižší než 100 mm/s. Při tuhém zastavení při vyšších rychlostech se některé osy mohou posunout nepředpověditelným směrem.

`[ \PStop ]`

Path Stop

Datový typ: `switch`

Pohyb robotu je zastaven tak rychle, jak je to možné, bez podržení TCP na dráze (měkké zastavení), když se hodnota vyhledávacího signálu změní na aktivní nebo když se hodnota perzistentní proměnné změní na TRUE. Robot je posunut na určitou vzdálenost, potom se zastaví a není posunut zpět k hledané pozici, tj. k pozici, kde se hodnoty signálu nebo perzistentní proměnné změnily.

`[ \SStop ]`

Soft Stop

Datový typ: `switch`

Pohyb robotu je zastaven tak rychle, jak je to možné a TCP je podrženo poblíž nebo na dráze (měkké zastavení), když se hodnota vyhledávacího signálu změní na aktivní nebo když se hodnota perzistentní proměnné změní na TRUE. Robot je posunut pouze na malou vzdálenost, potom se zastaví a není posunut zpět k hledané pozici, tj. k pozici, kde se signál změnil. `SStop` je rychlejší než `PStop`. Ale když robot běží rychleji než 100 mm/s, zastaví se ve směru tečny pohybu, což způsobí jeho okrajový skluz z dráhy.

`[ \Sup ]`

Supervision

Datový typ: `switch`

Vyhledávací instrukce je citlivá na aktivaci signálu nebo změnu hodnoty perzistentní proměnné během kompletního pohybu (flying search), tj. i po hlášené první změně signálu nebo změně hodnoty perzistentní proměnné. Jestliže se během hledání objeví více než jedna shoda, potom je vyvolána obnovitelná chyba s robotem v `ToPoint`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je vypuštěn argument `\Stop`, `\PStop`, `\SStop`, nebo `\Sup` (není použit žádný přepínač):

- pohyb pokračuje (flying search) k pozici určené v argumentu `ToPoint` (stejné jako u argumentu `\Sup`)
- chyba je hlášena kvůli žádnému záchytu hledání, ale není hlášena kvůli více než jednomu záchytu hledání (první záchyt hledání je vrácen jako `SearchPoint`)

`Signal`

Datový typ: `signaldi`

Jméno signálu, který bude pod dohledem.

`PersBool`

Datový typ: `bool`

Perzistentní proměnná, která bude pod dohledem.

`[ \Flanks ]`

Datový typ: `switch`

Kladná a záporná hrana signálu je platná pro záchyt hledání. Při použití argumentu `PersBool` je to změna hodnoty proměnné, která je platná pro záchyt hledání.

Pro signál: Jestliže je vypuštěn argument `\Flanks` pouze kladná hrana signálu je platná pro záchyt hledání a dohled signálu bude aktivován na začátku procesu hledání. To znamená, když signál má kladnou hodnotu již na začátku procesu hledání nebo komunikace se signálem je ztracena při zastavení pohybu robotu tak rychle, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_SIGSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

Pro perzistentní proměnnou: Jestliže je vypuštěn argument `\Flanks`, je to pouze v případě změny hodnoty na TRUE, tj. platný záchyt hledání a dohled nad proměnnou budou aktivovány na začátku procesu hledání. To znamená, když perzistentní proměnná má kladnou hodnotu již na začátku procesu hledání, potom je zastavení pohybu robotu tak rychlé, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_PERSSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

`[ \PosFlank ]`

Datový typ: `switch`

Kladná hrana signálu je platná pro záchyt hledání, nebo změna hodnoty na TRUE při použití perzistentní proměnné.

`[ \NegFlank ]`

Datový typ: `switch`

Záporná hrana signálu je platná pro záchyt hledání, nebo změna hodnoty na FALSE při použití perzistentní proměnné.

`[ \HighLevel ]`

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.216 SearchC - Hledá kruhově pomocí robotu
RobotWare - OS
Pokračování

Stejná funkčnost jako při nepoužívání přepínače `\Flanks`.

Pro signál: Kladná hrana signálu je platná pro záchyt hledání a dohled signálu bude aktivován na začátku procesu hledání. To znamená, když signál má kladnou hodnotu již na začátku procesu hledání nebo komunikace se signálem je ztracena, potom je zastavení pohybu robotu tak rychlé, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_SIGSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

Pro perzistentní proměnnou: Pouze v případě změny hodnoty na TRUE je platný záchyt hledání a dohled nad proměnnou bude aktivován na začátku procesu hledání. To znamená, když perzistentní proměnná má kladnou hodnotu již na začátku procesu hledání, potom je zastavení pohybu robotu tak rychlé, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_PERSSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

`[ \LowLevel ]`

Datový typ: `switch`

Pro signál: Záporná hrana signálu je platná pro záchyt hledání a dohled signálu bude aktivován na začátku procesu hledání. To znamená, když signál má hodnotu 0 již na začátku procesu hledání nebo komunikace se signálem je ztracena, potom je zastavení pohybu robotu tak rychlé, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_SIGSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

Pro perzistentní proměnnou: Pouze v případě změny hodnoty na FALSE je platný záchyt hledání a dohled nad proměnnou bude aktivován na začátku procesu hledání. To znamená, když perzistentní proměnná má hodnotu FALSE již na začátku procesu hledání, potom je zastavení pohybu robotu tak rychlé, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_PERSSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

`SearchPoint`

Datový typ: `robtarget`

Pozice TCP a externích os, když byl spuštěn vyhledávací signál. Pozice je určena v nejvzdálenějším souřadném systému a bere v úvahu určený nástroj, pracovní objekt a aktivní `ProgDisp/ExtOffs` souřadný systém into consideration.

`CirPoint`

Datový typ: `robtarget`

Kruhový bod robotu. Viz instrukce `MoveC`, kde je podrobnější popis kruhového pohybu. Kruhový bod je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`ToPoint`

Datový typ: `robtarget`

Bod určení (destinace) robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci). `SearchC` používá vždy stop bod jako zónová data pro destinaci.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \V ]`

Velocity

Datový typ: `num`

Tento argument se používá k určení rychlosti TCP v mm/sek. přímo v instrukci. Je potom vyměněn za odpovídající rychlost, určenou v rychlostních datech.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určené pozice (destinace).

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadný systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být stanoven pro lineární pohyb ve vztahu k pracovnímu objektu, který bude proveden.

`[ \Corr ]`

Correction

Datový typ: `switch`

Když je tento argument přítomen, korekční data zapsaná do vstupu korekcí instrukcí `CorrWrite` budou přidána k pozici dráhy a destinace.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.216 SearchC - Hledá kruhově pomocí robotu

RobotWare - OS

Pokračování

```
[ \TLoad ]
```

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

V instrukci `MoveC` najdete více informací o kruhovém pohybu.

Pohyb je vždy zakončen stop bodem, tj. robot se zastavuje v bodě destinace.

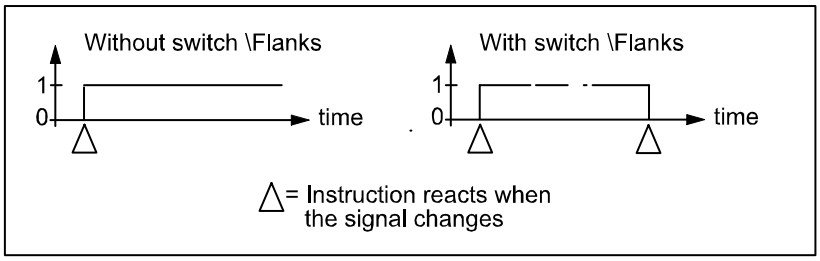
Když se používá průjezdné (flying) hledání, tzn. že je určen argument `\Sup` nebo není určen žádný přepínač, pohyb robotu vždy pokračuje k naprogramovanému bodu destinace. Když je hledání provedeno pomocí přepínače `\Stop`, `\PStop`, nebo `\SStop`, pohyb robotu se zastaví při zjištění prvního záchytu hledání.

Instrukce `SearchC` vrací pozici TCP, když se hodnota digitálního signálu nebo perzistentní proměnné změní na požadovanou hodnotu, jak je uvedeno na obrázku dole.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Obrázek ukazuje, jak se používá detekce bočně spouštěného signálu (pozice je uložena pouze při první změně signálu).



xx0500002237

Další příklady

Více příkladů jak používat instrukci `SearchC` je názorně uvedeno dole.

Příklad 1

```
SearchC \Sup, di1\Flanks, sp, cirpoint, p10, v100, probe;
```

TCP `probe` je v pohybu kruhově k pozici `p10`. Když se hodnota signálu `di1` změní na aktivní nebo pasivní, pozice se uloží do `sp`. Jestliže se hodnota signálu změní dvakrát, potom program vyvolá chybu.

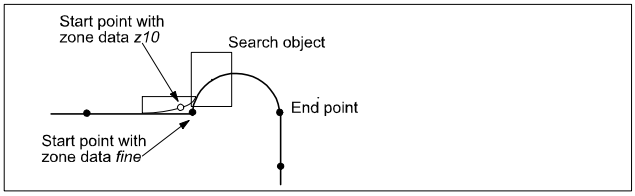
Omezení

Všeobecná omezení podle instrukce `MoveC`

Zónová data pro polohovací instrukci, která předchází `SearchC`, se musí používat opatrně. Začátek hledání, tj. když I/O signál je připraven reagovat, není v tomto případě naprogramovaný bod destinace předchozí polohovací instrukce, ale bod podél reálné dráhy robotu. Obrázek dole ilustruje příklad něčeho, co může skončit špatně, když jsou použita jiná zónová data než `fine`.

Instrukce `SearchC` by nikdy neměla být restartována po přejetí kruhového bodu. Jinak robot nevezme naprogramovanou dráhu (polohování kolem kruhové dráhy v jiném směru v porovnání s naprogramovanou).

Obrázek ukazuje, jak je shoda udělána na špatné straně objektu, protože byla použita špatná zónová data.



xx0500002238

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.216 SearchC - Hledá kruhově pomocí robotu
RobotWare - OS
Pokračování

VAROVÁNÍ

Omezení pro hledání, jestliže koordinované synchronizované pohyby:

- Při používání `SearchL`, `SearchC` nebo `SearchExtJ` pro jednu programovou úlohu a některou jinou pohybovou instrukci v jiné programové úloze je možné použít pouze průjezdové (flying) hledání s přepínačem `\Sup`. Kromě toho je jedině možné provádět obnovu po chybě s `TRYNEXT`.
- Je možné využívat veškeré vyhledávací funkčnosti, jestliže použijeme některou z instrukcí `SearchL`, `SearchC` nebo `SearchExtJ` ve všech zúčastněných programových úlohách s koordinovanými synchronizovanými pohyby a generovat nález ze stejného digitálního vstupního signálu. Toto generuje nález synchronně ve všech vyhledávacích instrukcích. Každá obnova po chybě musí být také stejná ve všech zúčastněných programových úlohách.

Když je hledání aktivní, není možné ukládat aktuální dráhu s instrukcí `StorePath`.

Přesnost opakování hledání pozice nálezu s rychlostí TCP 20 - 1000 mm/s 0,1 - 0,3 mm.

Typická stop vzdálenost pomocí rychlosti hledání 50 mm/s:

- bez TCP na dráze (přepínač `\Stop`) 1-3 mm
- s TCP na dráze (přepínač `\PStop`) 15-25 mm
- s TCP poblíž dráhy (přepínač `\SStop`) 4-8 mm

Omezení pro hledání na dopravníku:

- hledání zastaví robot při nálezu nebo když hledání selže, takže provádějte hledání ve stejném směru s pohybem dopravníku a pokračujte po zastavení hledání pohybem do bezpečné pozice. Pro pohyb do bezpečné pozice použijte ošetřování chyb, když hledání selže.
- přesnost opakování pro pozici nálezu bude horší, když se bude hledat na dopravníku, a závisí na rychlosti dopravníku a na stabilitě rychlosti.

Řešení chyb

Chyba je hlášena během hledání, když:

- proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO` - toto generuje chybu `ERR_NO_ALIAS_DEF`.
- neobjevilo se žádné zjištění signálu - toto generuje chybu `ERR_WHLSEARCH`.
- objevilo se zjištění více než jednoho signálu - toto generuje chybu `ERR_WHLSEARCH` pouze když je použit argument `\Sup`.
- signál má již kladnou hodnotu na začátku procesu hledání nebo komunikace se signálem je ztracena. Toto generuje chybu `ERR_SIGSUPSEARCH` pouze když je vypuštěn argument `\Flanks`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

- hodnota perzistentní proměnné je TRUE na začátku procesu hledání - toto generuje chybu ERR_PERSSUPSEARCH pouze když je vypuštěn argument \Flanks.

Chyby je možné ošetřovat různými způsoby podle zvoleného režimu běhu:

- **Nepřetržitý dopředu** / **Instrukce dopředu** / ERR_WHLSEARCH: Žádná pozice není vrácena a pohyb vždy pokračuje k naprogramovanému bodu destinace. Systémová proměnná ERRNO je nastavena na ERR_WHLSEARCH a chyba může být ošetřena v chybovém handleru rutiny.
- **Nepřetržitý dopředu** / **Instrukce dopředu** / ERR_SIGSUPSEARCH a ERR_PERSSUPSEARCH: Žádná pozice není vrácena a pohyb se vždy zastaví tak rychle jak je to možné na začátku dráhy hledání. Systémová proměnná ERRNO je nastavena na ERR_SIGSUPSEARCH nebo ERR_PERSSUPSEARCH podle použitého argumentu (signál nebo perzistentní proměnná) a chyba může být ošetřena v chybovém handleru rutiny.
- **Instrukce dozadu**: Během zpětného vykonávání provádí instrukce pohyb bez jakéhokoliv dohledu.

Syntaxe

```
SearchC
  [ ’\’ Stop ’,’] | [ ’\’ PStop ’,’] | [ ’\’ SStop ’,’] | [ ’\’
       Sup ’,’]
  [ Signal’:=’ ] < variable (VAR) of signaldi > |
  [ PersBool ':=' ] < persistent (PERS) of bool >
  [‘\’ Flanks] |
  [‘\’ PosFlank] |
  [‘\’ NegFlank] |
  [‘\’ HighLevel] |
  [‘\’ LowLevel] ’,’
  [ SearchPoint’:=’ ] < var or pers (INOUT) of robtarget > ’,’
  [ CirPoint’:=’ ] < expression (IN) of robtarget > ’,’
  [ ToPoint’:=’ ] < expression (IN) of robtarget > ’,’
  [ ’\’ ID ’:=’ < expression (IN) of identno >]’,’
  [ Speed’:=’ ] < expression (IN) of speeddata >
  [ ’\’ V ’:=’ < expression (IN) of num > ]|
  [ ’\’ T ’:=’ < expression (IN) of num > ] ’,’
  [ Tool ’:=’ ] < persistent (PERS) of tooldata >
  [ ’\’ WObj’:=’ < persistent (PERS) of wobjdata > ]
  [ ’\’ Corr ]
  [ ’\’ TLoad’ :=’ < persistent (PERS) of loaddata > ] ’;’
```

Související informace

Pro informace o	Viz
Lineární hledání	*SearchL - Hledá lineárně pomocí robotu na str 603*
Zapisuje do vstupu korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
Pohybuje robotem kruhově	*MoveC - Pohybuje robotem kruhově na str 358*
Kruhový pohyb	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.216 SearchC - Hledá kruhově pomocí robotu

RobotWare - OS

Pokračování

Pro informace o	Viz
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Používání chybových handlerů	*Technická referenční příručka - Přehled RAPID*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.217 SearchExtJ - Hledat s jednou nebo několika mechanickými jednotkami bez TCP

Použití

`SearchExtJ` (*Search External Joints*) se používá pro hledání pozic externích os při posunování pouze lineárních nebo rotačních externích os. Externí osy mohou patřit jedné nebo několika mechanickým jednotkám bez TCP.

Během pohybu robot dohlíží na digitální vstupní signál nebo perzistentní proměnnou. Když se hodnota signálu nebo perzistentní proměnné mění na požadovanou hodnotu, robot okamžitě přečte aktuální pozici.

Tuto instrukci je možné použít pouze když:

- Aktuální programová úloha je definována jako pohybová úloha
- Úloha kontroluje jednu nebo několik mechanických jednotek bez TCP

Při používání vyhledávacích instrukcí je důležité konfigurovat I/O systém, abychom měli velmi krátkou časovou prodlevu od nastavení fyzického signálu k okamžiku, kdy systém dostane informace o nastavení (použijte jednotku I/O s kontrolou přerušení, nikoliv kontrolu volby). Způsob, jak to udělat, se může mezi aplikačními sběrnicemi lišit. Při používání DeviceNet potom ABB jednotky DSQC 651 (AD Combi I/O) a DSQC 652 (Digital I/O) budou dávat krátké časové prodlevy, jelikož používají spojovací typ Změna stavu. Při použití jiných aplikačních sběrnic zajistěte konfiguraci sítě řádným způsobem, aby byly získány správné podmínky.

Základní příklady

Následující příklady názorně ukazují instrukci `SearchExtJ`:

Viz také *Další příklady na str 599*.

Příklad 1

```
SearchExtJ di1, searchp, jpos10, vrot20;
```

Mechanická jednotka s rotačními osami je v pohybu k pozici `jpos10`rychlostí `vrot20`. Když se hodnota signálu `di1`změní na aktivní, pozice se uloží do `searchp`.

Příklad 2

```
SearchExJ \Stop, di2, posx, jpos20, vlin50;
```

Mechanická jednotka s lineární osou je v pohybu kruhově k pozici `jpos20`. Když se hodnota signálu `di2`změní na aktivní, pozice se uloží do `posx` a probíhající pohyb se okamžitě zastaví.

Příklad 3

```
PERS bool mypers:=FALSE;
...
SearchExJ \Stop, di2, posx, jpos20, vlin50;
```

Mechanická jednotka s lineární osou je v pohybu k pozici `jpos20`. Když se hodnota perzistentní proměnné `mypers`změní na TRUE, pozice se uloží do `posx` a probíhající pohyb se okamžitě zastaví.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.217 SearchExtJ - Hledat s jednou nebo několika mechanickými jednotkami bez TCP

RobotWare - OS

Pokračování

Argumenty

```
SearchExtJ [\Stop] | [\PStop] | [\SStop] | [\Sup] Signal | PersBool
     [\Flanks] | [\PosFlank] | [\NegFlank] | [\HighLevel] |
     [\LowLevel] SearchJointPos ToJointPos [\ID] [\UseEOffs] Speed
     [\T]
```

`[ \Stop ]`

Stiff Stop

Datový typ: `switch`

Pohyb je zastaven tak rychle, jak je to možné (tvrdé zastavení), když se hodnota vyhledávacího signálu změní na aktivní nebo když se hodnota perzistentní proměnné změní na TRUE. Externí osy jsou posunuty na malou vzdálenost, potom se zastaví a nejsou posunuty zpět k hledané pozici, tj. k pozici, kde se signál změnil.

`[ \PStop ]`

Path Stop

Datový typ: `switch`

Pohyb je zastaven dráhovým zastavením (Program Stop), když se hodnota vyhledávacího signálu změní na aktivní nebo když se hodnota perzistentní proměnné změní na TRUE. Externí osy jsou posunuty na poměrně velkou vzdálenost, potom se zastaví a nejsou posunuty zpět k hledané pozici, tj. k pozici, kde se signál změnil.

`[ \SStop ]`

Soft Stop

Datový typ: `switch`

Pohyb je zastaven tak rychle, jak je to možné rychlým měkkým zastavením, když se hodnota vyhledávacího signálu změní na aktivní nebo když se hodnota perzistentní proměnné změní na TRUE. Externí osy jsou posunuty pouze o malou vzdálenost, potom se zastaví a nejsou posunuty zpět k hledané pozici, tj. k pozici, kde se signál změnil.

`Stop` je rychlejší v porovnáním s `SStop`. `SStop` je rychlejší v porovnáním s `PStop`.

`[ \Sup ]`

Supervision

Datový typ: `switch`

Vyhledávací instrukce je citlivá na aktivaci signálu nebo změnu hodnoty perzistentní proměnné během kompletního pohybu (flying search), tj. i po hlášené první změně signálu nebo změně hodnoty perzistentní proměnné. Jestliže se během hledání objeví více než jedna shoda, potom je vyvolána obnovitelná chyba s robotem v `ToPoint`.

Jestliže je vypuštěn argument `\Stop`, `\PStop`, `\SStop`, nebo `\Sup` je vypuštěn (není použit žádný přepínač):

- Pohyb pokračuje (flying search) k pozici určené v argumentu `ToJointPos` (stejné jako u argumentu `\Sup`)
- Chyba je hlášena kvůli jednomu nálezu, ale není hlášena kvůli více než jednomu nálezu (první nález je vrácen jako `SearchJointPos`)

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Signal`

Datový typ: `signaldi`

Jméno signálu, který bude pod dohledem.

`PersBool`

Datový typ: `bool`

Perzistentní proměnná, která bude pod dohledem.

`[ \Flanks ]`

Datový typ: `switch`

Kladná a záporná hrana signálu je platná pro záchyt hledání. Při použití argumentu `PersBool` **je to změna hodnoty proměnné, která je platná pro záchyt hledání.**

Pro signál: Jestliže je vypuštěn argument `\Flanks`**, pouze kladná hrana signálu je platná pro nález a dohled signálu bude aktivován na začátku procesu hledání. To znamená, když signál má kladnou hodnotu již na začátku procesu hledání nebo komunikace se signálem je ztracena, pohyb je zastaven tak rychle, jak je to možné. Chyba s uživatelskou obnovou** `ERR_SIGSUPSEARCH` **bude vyvolána a může být ošetřena v chybovém handleru.**

Pro perzistentní proměnnou: Jestliže je vypuštěn argument `\Flanks`**, je to pouze v případě změny hodnoty na TRUE, tj. platný nález a dohled nad proměnnou budou aktivovány na začátku procesu hledání. To znamená, když perzistentní proměnná má kladnou hodnotu již na začátku procesu hledání, potom je zastavení pohybu tak rychlé, jak je to možné. Chyba s uživatelskou obnovou** `ERR_PERSSUPSEARCH` **bude vyvolána a může být ošetřena v chybovém handleru.**

`[ \PosFlank ]`

Datový typ: `switch`

Kladná hrana signálu je platná pro záchyt hledání, nebo změna hodnoty na TRUE při použití perzistentní proměnné.

`[ \NegFlank ]`

Datový typ: `switch`

Záporná hrana signálu je platná pro záchyt hledání, nebo změna hodnoty na FALSE při použití perzistentní proměnné.

`[ \HighLevel ]`

Datový typ: `switch`

Stejná funkčnost jako při nepoužívání přepínače `\Flanks`**.**

Pro signál: Kladná hrana signálu je platná pro nález a dohled signálu bude aktivován na začátku procesu hledání. To znamená, když signál má kladnou hodnotu již na začátku procesu hledání nebo komunikace se signálem je ztracena, pohyb je zastaven tak rychle, jak je to možné. Chyba s uživatelskou obnovou `ERR_SIGSUPSEARCH` **bude vyvolána a může být ošetřena v chybovém handleru.**

Pro perzistentní proměnnou: Pouze změna hodnoty na TRUE je platný nález a dohled nad proměnnou bude aktivován na začátku procesu hledání. To znamená, když perzistentní proměnná má kladnou hodnotu již na začátku procesu hledání, potom je zastavení pohybu tak rychlé, jak je to možné. Chyba s uživatelskou

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.217 SearchExtJ - Hledat s jednou nebo několika mechanickými jednotkami bez TCP
RobotWare - OS
Pokračování

obnovou ERR_PERSSUPSEARCH bude vyvolána a může být ošetřena v chybovém handleru.

[\LowLevel]

Datový typ: switch

Pro signál: Kladná hrana signálu je platná pro nález a dohled signálu bude aktivován na začátku procesu hledání. To znamená, když signál má hodnotu 0 již na začátku procesu hledání nebo komunikace se signálem je ztracena, pohyb je zastaven tak rychle, jak je to možné. Chyba s uživatelskou obnovou ERR_SIGSUPSEARCH bude vyvolána a může být ošetřena v chybovém handleru.

Pro perzistentní proměnnou: Pouze změna hodnoty na FALSE je platný nález a dohled nad proměnnou bude aktivován na začátku procesu hledání. To znamená, když perzistentní proměnná má hodnotu FALSE již na začátku procesu hledání, potom je zastavení pohybu tak rychlé, jak je to možné. Chyba s uživatelskou obnovou ERR_PERSSUPSEARCH bude vyvolána a může být ošetřena v chybovém handleru.

SearchJointPos

Datový typ: jointtarget

Pozice externích os při spuštění vyhledávacího signálu. Pozice bere v úvahu všechny aktivní ExtOffs

ToJointPos

Datový typ: jointtarget

Bod určení (destinace) pro externí osy. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci). SearchExtJ používá vždy stop bod jako zónová data pro destinaci.

[\ID]

Synchronization id

Datový typ: identno

Argument [\ID] je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

[\UseEOffs]

Use External Offset

Datový typ: switch

Ofset pro externí osy, nastavený instrukcí EOffsSet, je aktivován pro instrukci SearchExtJ, když je použit argument UseEOffs. Více informací o externím ofsetu najdete v instrukci EOffsSet.

Speed

Datový typ: speeddata

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost lineárních nebo rotačních externích os.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se pohybují mechanické jednotky. Je potom vyměněn za odpovídající rychlostní data.

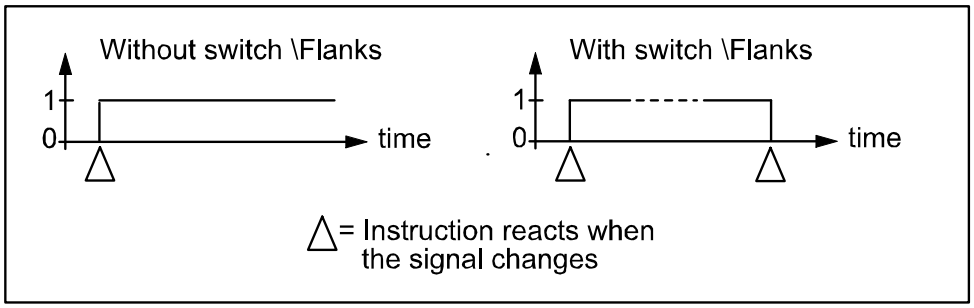
Vykonávání programu

Viz instrukce `MoveExtJ`, kde jsou informace o pohybu mechanických jednotek bez TCP.

Pohyb končí vždy se stop bodem, tj. externí osy se zastaví na bodu destinace. Jestliže se používá průjezdné (flying) hledání, tj. argument `\Sup` je určen, ale není určen žádný přepínač, pohyb vždy pokračuje k naprogramovanému bodu destinace. Jestliže je hledání prováděno pomocí přepínače `\Stop`, `\PStop` nebo `\SStop`, pohyb se zastaví, když je zjištěn první nález.

Instrukce `SearchExtJ` ukládá pozici externích os, když se hodnota digitálního signálu nebo perzistentní proměnné změní na požadovanou hodnotu, jak je uvedeno na obrázku dole.

Obrázek ukazuje, jak se používá detekce bočně spouštěného signálu (pozice je uložena pouze při první změně signálu).



xx0500002243

Další příklady

Více příkladů jak používat instrukci `SearchExtJ` je názorně uvedeno dole.

Příklad 1

```
SearchExtJ \Sup, di1\Flanks, searchp,jpos10, vrot20;
```

Mechanická jednotka je v pohybu k pozici `jpos10`. Když se hodnota signálu `di1` změní na aktivní nebo pasivní, pozice se uloží do `searchp`. Jestliže se hodnota signálu změní dvakrát, potom program vyvolá chybu po dokončení procesu hledání.

Příklad 2

```
SearchExtJ \Stop, di1, sp, jpos20, vlin50;
MoveExtJ sp, vlin50, fine \Inpos := inpos50;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.217 SearchExtJ - Hledat s jednou nebo několika mechanickými jednotkami bez TCP
RobotWare - OS
Pokračování

Kontrola na signálu `dil` bude provedena na začátku procesu hledání, a jestliže signál již má kladnou hodnotu nebo komunikace se signálem je ztracena, pohyb se zastaví. Jinak je mechanická jednotka posunována k pozici `jpos20`. Když se hodnota signálu `di1` změní na aktivní, pozice je uložena do `sp`. Mechanická jednotka je posunuta zpět k tomuto bodu pomocí přesně definovaného stop bodu.

Řešení chyb

Chyba je hlášena během hledání, když:

- proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO` - toto generuje chybu `ERR_NO_ALIASIO_DEF`.
- Neobjevilo se žádné zjištění signálu - toto generuje chybu `ERR_WHLSEARCH`.
- Objevilo se zjištění více než jednoho signálu - toto generuje chybu `ERR_WHLSEARCH` pouze když je použit argument `\Sup`.
- Signál má již kladnou hodnotu na začátku procesu hledání nebo komunikace se signálem je ztracena. Toto generuje chybu `ERR_SIGSUPSEARCH` pouze když je vypuštěn argument `\Flanks`.
- hodnota perzistentní proměnné je TRUE na začátku procesu hledání - toto generuje chybu `ERR_PERSSUPSEARCH` pouze když je vypuštěn argument `\Flanks`.

Chyby je možné ošetřovat různými způsoby podle zvoleného režimu běhu:

- **Nepřetržitý dopředu** / **Instrukce dopředu** / `ERR_WHLSEARCH`: Žádná pozice není vrácena a pohyb vždy pokračuje k naprogramovanému bodu destinace. Systémová proměnná `ERRNO` je nastavena na `ERR_WHLSEARCH` a chyba může být ošetřena v chybovém handleru rutiny.
- **Nepřetržitý dopředu** / **Instrukce dopředu** / `ERR_SIGSUPSEARCH` a `ERR_PERSSUPSEARCH`: Žádná pozice není vrácena a pohyb se vždy zastaví tak rychle jak je to možné na začátku dráhy hledání. Systémová proměnná `ERRNO` je nastavena na `ERR_SIGSUPSEARCH` nebo `ERR_PERSSUPSEARCH` podle použitého argumentu (signál nebo perzistentní proměnná) a chyba může být ošetřena v chybovém handleru rutiny.
- **Instrukce dozadu**: Během zpětného vykonávání provádí instrukce pohyb bez jakéhokoliv dohledu.

Příklad

```
VAR num fk;
...
MoveExtJ jpos10, vrot100, fine;
SearchExtJ \Stop, di1, sp, jpos20, vrot5;
...
ERROR
  IF ERRNO=ERR_WHLSEARCH THEN
    StorePath;
    MoveExtJ jpos10, vrot50, fine;
    RestoPath;
    ClearPath;
    StartMove;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
    RETRY;
  ELSEIF ERRNO=ERR_SIGSUPSEARCH THEN
    TPWrite "The signal of the SearchExtJ instruction is already
        high!";
    TPReadFK fk,"Try again after manual reset of signal ?","YES",
        stEmpty, stEmpty, stEmpty, "NO";
    IF fk = 1 THEN
      StorePath;
      MoveExtJ jpos10, vrot50, fine;
      RestoPath;
      ClearPath;
      StartMove;
      RETRY;
    ELSE
      Stop;
    ENDIF
  ENDIF
```

Jestliže signál je již aktivní na začátku procesu hledání nebo komunikace se signálem je ztracena, bude aktivován uživatelský dialog (`TPReadFK ...;`). Resetujte signál a stiskněte YES na uživatelském dialogu a mechanická jednotka se posune zpět k `jpos10` a provede další pokus. Jinak bude vykonávání programu zastaveno.

Jestliže signál je pasivní na začátku procesu hledání, mechanická jednotka bude hledat od pozice `jpos10` k `jpos20`. Jestliže se neobjeví žádné zjištění signálu, robot se posune zpět k `jpos10` a provede další pokus.

Omezení

Omezení pro hledání, jestliže koordinované synchronizované pohyby:

- Při používání `SearchL`, `SearchC` nebo `SearchExtJ` pro jednu programovou úlohu a některou jinou pohybovou instrukci v jiné programové úloze je možné použít pouze průjezdové (flying) hledání s přepínačem `\Sup`. Kromě toho je jedině možné provádět obnovu po chybě s `TRYNEXT`.
- Je možné využívat veškeré vyhledávací funkčnosti, jestliže použijeme některou z instrukcí `SearchL`, `SearchC` nebo `SearchExtJ` ve všech zúčastněných programových úlohách s koordinovanými synchronizovanými pohyby a generovat nálezy ze stejného digitálního vstupního signálu. Toto generuje nálezy synchronně ve všech vyhledávacích instrukcích. Každá obnova po chybě musí být také stejná ve všech zúčastněných programových úlohách.
- Když je hledání aktivní, není možné ukládat aktuální dráhu s instrukcí `StorePath`.

Syntaxe

```
SearchExtJ
  [ '\' Stop ',' ] | [ '\' PStop ',' ] | [ '\' SStop ',' ] | [ '\'
      Sup ',' ]
  [ Signal ':=' ] < variable (VAR) of signaldi > |
  [ PersBool ':=' ] < persistent (PERS) of bool >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.217 SearchExtJ - Hledat s jednou nebo několika mechanickými jednotkami bez TCP

RobotWare - OS

Pokračování

```
[‘\’ Flanks] |
[‘\’ PosFlank] |
[‘\’ NegFlank] |
[‘\’ HighLevel] |
[‘\’ LowLevel] ’,’
[ SearchJointPos’ :=’ ] < var or pers (INOUT) of jointtarget >
     ’,’
[ ToJointPos’ :=’ ] < expression (IN) of jointtarget >
[ ’\’ ID ’:=’ < expression (IN) of identno >]’,’
[ ’\’ UseEOffs’ ,’ ]
[ Speed ’:=’ ] < expression (IN) of speeddata >
[ ’\’ T ’:=’ < expression (IN) of num > ] ’;’
```

Související informace

Pro informace o	Viz
Posunout mechanické jednotky bez TCP	*MoveExtJ - Uvést do pohybu jednu nebo několik mechanických jednotek bez TCP na str 385*
Definice jointtarget	*jointtarget - Data pozice svaru na str 1520*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Používání chybových handlerů	*Technická referenční příručka - Přehled RAPID*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.218 SearchL - Hledá lineárně pomocí robotu

Použití

`SearchL` (*Search Linear*) se používá pro vyhledání pozice při lineárním pohybu středním bodem nástroje (TCP).

Během pohybu robot dohlíží na digitální vstupní signál nebo perzistentní proměnnou. Když se hodnota signálu nebo perzistentní proměnné mění na požadovanou hodnotu, robot okamžitě přečte aktuální pozici.

Tuto instrukci je možné používat typicky, když nástroj držený robotem je sonda pro detekci povrchu. Vnější souřadnice pracovního objektu je možné získat pomocí instrukce `SearchL`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Při používání vyhledávacích instrukcí je důležité konfigurovat I/O systém, abychom měli velmi krátký čas od nastavení fyzického signálu k systému a dostali tak informace o nastavení (použijte jednotku I/O s kontrolou přerušení, nikoliv kontrolu volby). Způsob, jak to udělat, se může mezi aplikačními sběrnicemi lišit. Při používání DeviceNet potom ABB jednotky DSQC 651 (AD Combi I/O) a DSQC 652 (Digital I/O) budou dávat krátké časy, jelikož používají spojovací typ Změna stavu. Při použití jiných aplikačních sběrnic zajistěte konfiguraci sítě řádným způsobem, aby byly získány správné podmínky.

Základní příklady

Následující příklady názorně ukazují instrukci `SearchL`:

Viz také *Další příklady na str 609*.

Příklad 1

```
SearchL di1, sp, p10, v100, probe;
```

TCP `probe` je v pohybu lineárně k pozici `p10` rychlostí `v100`. Když se hodnota signálu `di1`změní na aktivní, pozice se uloží do `sp`.

Příklad 2

```
SearchL \Stop, di2, sp, p10, v100, probe;
```

TCP `probe` je v pohybu lineárně k pozici `p10`. Když se hodnota signálu `di2`změní na aktivní, pozice se uloží do `sp` a robot se okamžitě zastaví.

Příklad 3

```
PERS bool mypers:=FALSE;
...
SearchL mypers, sp, p10, v100, probe;
```

TCP `probe` je v pohybu lineárně k pozici `p10` rychlostí `v100`. Když se hodnota perzistentní proměnné `mypers`změní na TRUE, pozice se uloží do `sp`.

Argumenty

```
SearchL [\Stop] | [\PStop] | [\SStop] | [\Sup] Signal | PersBool
     [\Flanks] | [\PosFlank] | [\NegFlank] | [\HighLevel] |
     [\LowLevel] SearchPoint ToPoint [\ID] Speed [\V] | [\T] Tool
     [\WObj] [\Corr] [\TLoad]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.218 SearchL - Hledá lineárně pomocí robotu
RobotWare - OS
Pokračování

`[ \Stop ]`

Stiff Stop

Datový typ: `switch`

Pohyb robotu je zastaven tak rychle, jak je to možné, bez podržení TCP na dráze (tvrdé zastavení), když se hodnota vyhledávacího signálu změní na aktivní nebo když se hodnota perzistentní proměnné změní na TRUE. Robot je posunut na malou vzdálenost, potom se zastaví a není posunut zpět k hledané pozici, tj. k pozici, kde se signál nebo perzistentní proměnná změnily.

VAROVÁNÍ

Zastavení hledání tuhým zastavením (přepínač `\Stop`) je dovoleno pouze v případě, že rychlost TCP je nižší než 100 mm/s. Při tuhém zastavení při vyšších rychlostech se některé osy mohou posunout nepředvídatelným směrem.

`[ \PStop ]`

Path Stop

Datový typ: `switch`

Pohyb robotu je zastaven tak rychle, jak je to možné, bez podržení TCP na dráze (měkké zastavení), když se hodnota vyhledávacího signálu změní na aktivní nebo když se hodnota perzistentní proměnné změní na TRUE. Robot je posunut na určitou vzdálenost, potom se zastaví a není posunut zpět k hledané pozici, tj. k pozici, kde se hodnoty signálu nebo perzistentní proměnné změnily.

`[ \SStop ]`

Soft Stop

Datový typ: `switch`

Pohyb robotu je zastaven tak rychle, jak je to možné a TCP je podrženo poblíž nebo na dráze (měkké zastavení), když se hodnota vyhledávacího signálu změní na aktivní nebo když se hodnota perzistentní proměnné změní na TRUE. Robot je posunut pouze na malou vzdálenost, potom se zastaví a není posunut zpět k hledané pozici, tj. k pozici, kde se signál nebo perzistentní proměnná změnily. `SStop` je rychlejší než `PStop`. Ale když robot běží rychleji než 100 mm/s, zastaví se ve směru tečny pohybu, což způsobí jeho okrajový skluz z dráhy.

`[ \Sup ]`

Supervision

Datový typ: `switch`

Vyhledávací instrukce je citlivá na aktivaci signálu nebo změnu hodnoty perzistentní proměnné během kompletního pohybu (flying search), tj. i po hlášené první změně signálu nebo změně hodnoty perzistentní proměnné. Jestliže se během hledání objeví více než jedna shoda, potom je vyvolána obnovitelná chyba s robotem v `ToPoint`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je vypuštěn argument `\Stop`, `\PStop`, `\SStop`, nebo `\Sup` (není použit žádný přepínač):

- pohyb pokračuje (flying search) k pozici určené v argumentu `ToPoint` (stejné jako u argumentu `\Sup`)
- chyba je hlášena kvůli žádnému záchytu hledání, ale není hlášena kvůli více než jednomu záchytu hledání (první záchyt hledání je vrácen jako `SearchPoint`)

`Signal`

Datový typ: `signaldi`

Jméno signálu, který bude pod dohledem.

`PersBool`

Datový typ: `bool`

Perzistentní proměnná, která bude pod dohledem.

`[ \Flanks ]`

Datový typ: `switch`

Kladná a záporná hrana signálu je platná pro záchyt hledání. Při použití argumentu `PersBool` je to změna hodnoty proměnné, která je platná pro záchyt hledání.

Pro signál: Jestliže je vypuštěn argument `\Flanks` pouze kladná hrana signálu je platná pro záchyt hledání a dohled signálu bude aktivován na začátku procesu hledání. To znamená, když signál má kladnou hodnotu již na začátku procesu hledání nebo komunikace se signálem je ztracena při zastavení pohybu robotu tak rychle, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_SIGSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

Pro perzistentní proměnnou: Jestliže je vypuštěn argument `\Flanks`, je to pouze v případě změny hodnoty na TRUE, tj. platný záchyt hledání a dohled nad proměnnou budou aktivovány na začátku procesu hledání. To znamená, když perzistentní proměnná má kladnou hodnotu již na začátku procesu hledání, potom je zastavení pohybu robotu tak rychlé, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_PERSSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

`[ \PosFlank ]`

Datový typ: `switch`

Kladná hrana signálu je platná pro záchyt hledání, nebo změna hodnoty na TRUE při použití perzistentní proměnné.

`[ \NegFlank ]`

Datový typ: `switch`

Záporná hrana signálu je platná pro záchyt hledání, nebo změna hodnoty na FALSE při použití perzistentní proměnné.

`[ \HighLevel ]`

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.218 SearchL - Hledá lineárně pomocí robotu
RobotWare - OS
Pokračování

Stejná funkčnost jako při nepoužívání přepínače `\Flanks`.

Pro signál: Kladná hrana signálu je platná pro záchyt hledání a dohled signálu bude aktivován na začátku procesu hledání. To znamená, když signál má kladnou hodnotu již na začátku procesu hledání nebo komunikace se signálem je ztracena, potom je zastavení pohybu robotu tak rychlé, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_SIGSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

Pro perzistentní proměnnou: Pouze v případě změny hodnoty na TRUE je platný záchyt hledání a dohled nad proměnnou bude aktivován na začátku procesu hledání. To znamená, když perzistentní proměnná má kladnou hodnotu již na začátku procesu hledání, potom je zastavení pohybu robotu tak rychlé, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_PERSSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

`[ \LowLevel ]`

Datový typ: `switch`

Pro signál: Záporná hrana signálu je platná pro záchyt hledání a dohled signálu bude aktivován na začátku procesu hledání. To znamená, když signál má hodnotu 0 již na začátku procesu hledání nebo komunikace se signálem je ztracena, potom je zastavení pohybu robotu tak rychlé, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_SIGSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

Pro perzistentní proměnnou: Pouze v případě změny hodnoty na FALSE je platný záchyt hledání a dohled nad proměnnou bude aktivován na začátku procesu hledání. To znamená, když perzistentní proměnná má hodnotu FALSE již na začátku procesu hledání, potom je zastavení pohybu robotu tak rychlé, jak je to možné, zatímco TCP je udržován na dráze (měkké zastavení). Chyba s uživatelskou obnovou `ERR_PERSSUPSEARCH` bude vyvolána a může být ošetřena v chybovém handleru.

`SearchPoint`

Datový typ: `robtarget`

Pozice TCP a externích os, když byl spuštěn vyhledávací signál. Pozice je určena v nejvzdálenějším souřadném systému a bere v úvahu určený nástroj, pracovní objekt a aktivní `ProgDisp/ExtOffs` souřadný systém.

`ToPoint`

Datový typ: `robtarget`

Bod určení (destinace) robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci). `SearchL` používá vždy stop bod jako zónová data pro destinaci.

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \V ]`

Velocity

Datový typ: `num`

Tento argument se používá k určení rychlosti TCP v mm/sek. přímo v instrukci. Je potom vyměněn za odpovídající rychlost, určenou v rychlostních datech.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určené pozice (destinace).

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být stanoven pro lineární pohyb ve vztahu k pracovnímu objektu, který bude proveden.

`[ \Corr ]`

Correction

Datový typ: `switch`

Korekční data zapsaná do vstupu korekcí instrukcí `CorrWrite` budou přidána k pozici dráhy a destinace, jestliže tento argument je přítomen.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.218 SearchL - Hledá lineárně pomocí robotu
RobotWare - OS
Pokračování

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

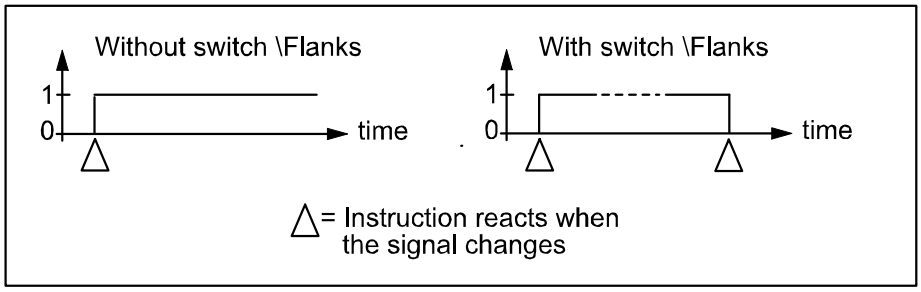
Vykonávání programu

V instrukci `MoveL` najdete více informací o lineárním pohybu.

Pohyb končí vždy se stop bodem, tj. robot se zastaví na bodu destinace. Jestliže se používá průjezdné (flying) hledání, tj. argument `\Sup` je určen nebo není určen žádný přepínač, pohyb robotu vždy pokračuje k naprogramovanému bodu destinace. Jestliže je hledání prováděno pomocí přepínače `\Stop`, `\PStop` nebo `\SStop`, pohyb robotu se zastaví, když je zjištěn první nález.

Instrukce `SearchL` ukládá pozici TCP, když se hodnota digitálního signálu nebo perzistentní proměnné změní na požadovanou hodnotu, jak je uvedeno na obrázku dole.

Obrázek ukazuje, jak se používá detekce bočně spouštěného signálu (pozice je uložena pouze při první změně signálu).



xx0500002243

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Více příkladů jak používat instrukci `SearchL` je názorně uvedeno dole.

Příklad 1

```
SearchL \Sup, di1 \Flanks, sp, p10, v100, probe;
```

TCP `probe` je v pohybu lineárně k pozici `p10`. Když se hodnota signálu `di1` změní na aktivní nebo pasivní, pozice se uloží do `sp`. Jestliže se hodnota signálu změní dvakrát, potom program vyvolá chybu po dokončení procesu hledání.

Příklad 2

```
SearchL \Stop, di1, sp, p10, v100, tool1;
MoveL sp, v100, fine \Inpos := inpos50, tool1;
PDispOn *, tool1;
MoveL p100, v100, z10, tool1;
MoveL p110, v100, z10, tool1;
MoveL p120, v100, z10, tool1;
PDispOff;
```

Na začátku procesu hledání bude provedena kontrola na signálu `di1`, a jestliže signál má již kladnou hodnotu nebo komunikace se signálem byla ztracena, robot se zastaví. Jinak je TCP `tool1` posunuto lineárně k pozici `p10`. Když se hodnota signálu `di1` změní na aktivní, pozice je uložena do `sp`.. Robot je posunut zpět k tomuto bodu pomocí přesně definovaného stop bodu. Pomocí posunu programu se robot potom pohybuje ve vztahu k hledané pozici, `sp`.

Příklad 3

```
PERS bool MyTrigger:=FALSE;
...
SearchL \Stop, MyTrigger, sp, p10, v100, tool1;
MoveL sp, v100, fine \Inpos := inpos50, tool1;
PDispOn *, tool1;
MoveL p100, v100, z10, tool1;
MoveL p110, v100, z10, tool1;
MoveL p120, v100, z10, tool1;
PDispOff;
```

Na začátku procesu hledání bude provedena kontrola na perzistentní proměnné `MyTrigger`, a jestliže proměnná je již TRUE, robot se zastaví. Jinak je TCP `tool1` posunuto lineárně k pozici `p10`. Když se hodnota perzistentní proměnné `MyTrigger` změní na TRUE, pozice je uložena do `sp`.. Robot je posunut zpět k tomuto bodu pomocí přesně definovaného stop bodu. Pomocí posunu programu se robot potom pohybuje ve vztahu k hledané pozici, `sp`.

Pokračování na další straně

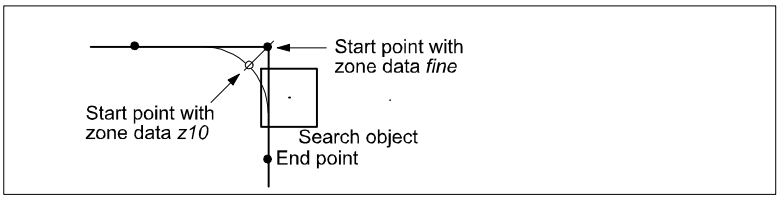
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.218 SearchL - Hledá lineárně pomocí robotu
RobotWare - OS
Pokračování

Omezení

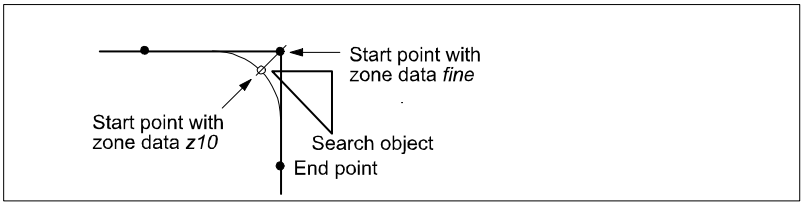
Zónová data pro polohovací instrukci, která předchází `SearchL`, se musí používat opatrně. Začátek hledání, tj. když I/O signál je připraven reagovat, není v tomto případě naprogramovaný bod destinace předchozí polohovací instrukce, ale bod podél reálné dráhy robotu. Obrázek dole ilustruje příklady situací, které mohou skončit špatně, když jsou použita jiná zónová data než `fine`.

Následující obrázek ukazuje, jak je shoda udělána na špatné straně objektu, protože byla použita špatná zónová data.



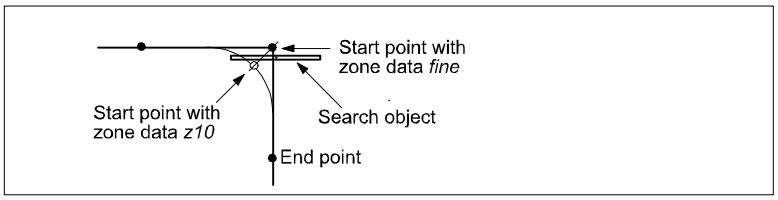
xx0500002244

Následující obrázek ukazuje, že nebyla zjištěna žádná shoda, protože byla použita špatná zónová data.



xx0500002245

Následující obrázek ukazuje, že nebyla zjištěna žádná shoda, protože byla použita špatná zónová data.



xx0500002246

Omezení pro hledání, jestliže koordinované synchronizované pohyby:

- Při používání `SearchL`, `SearchC` nebo `SearchExtJ` pro jednu programovou úlohu a některou jinou pohybovou instrukci v jiné programové úloze je možné použít pouze průjezdové (flying) hledání s přepínačem `\Sup`. Kromě toho je jedině možné provádět obnovu po chybě s `TRYNEXT`.
- Je možné využívat veškeré vyhledávací funkčnosti, jestliže použijeme některou z instrukcí `SearchL`, `SearchC`nebo `SearchExtJ` ve všech zúčastněných programových úlohách s koordinovanými synchronizovanými pohyby a generovat nález ze stejného digitálního vstupního signálu. Toto generuje nález synchronně ve všech vyhledávacích instrukcích. Každá obnova po chybě musí být také stejná ve všech zúčastněných programových úlohách.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Když je hledání aktivní, není možné ukládat aktuální dráhu s instrukcí `StorePath`.

Přesnost opakování hledání pozice nálezu s rychlostí TCP 20 - 1000 mm/s 0,1 - 0,3 mm.

Typická stop vzdálenost pomocí rychlosti hledání 50 mm/s:

- bez TCP na dráze (přepínač `\Stop`) 1-3 mm
- s TCP na dráze (přepínač `\PStop`) 15-25 mm
- s TCP poblíž dráhy (přepínač `\SStop`) 4-8 mm

Omezení pro hledání na dopravníku:

- hledání zastaví robot při nálezu nebo když hledání selže, takže provádějte hledání ve stejném směru s pohybem dopravníku a pokračujte po zastavení hledání pohybem do bezpečné pozice. Pro pohyb do bezpečné pozice použijte ošetřování chyb, když hledání selže.
- přesnost opakování pro pozici nálezu bude horší, když se bude hledat na dopravníku, a závisí na rychlosti dopravníku a na stabilitě rychlosti.

Řešení chyb

Chyba je hlášena během hledání, když:

- proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO` - toto generuje chybu `ERR_NO_ALIASIO_DEF`.
- neobjevilo se žádné zjištění - toto generuje chybu `ERR_WHLSEARCH`.
- objevilo se více než jedno zjištění - toto generuje chybu `ERR_WHLSEARCH`, pouze když je použit argument `\Sup`.
- signál má již kladnou hodnotu na začátku procesu hledání nebo komunikace se signálem je ztracena. Toto generuje chybu `ERR_SIGSUPSEARCH`, pouze když je vypuštěn argument `\Flanks`.
- hodnota perzistentní proměnné je TRUE na začátku procesu hledání - toto generuje chybu `ERR_PERSSUPSEARCH` pouze když je vypuštěn argument `\Flanks`.

Chyby je možné ošetřovat různými způsoby podle zvoleného režimu běhu:

- **Nepřetržitý dopředu** / **Instrukce dopředu** / `ERR_WHLSEARCH`: Žádná pozice není vrácena a pohyb vždy pokračuje k naprogramovanému bodu destinace. Systémová proměnná `ERRNO` je nastavena na `ERR_WHLSEARCH` a chyba může být ošetřena v chybovém handleru rutiny.
- **Nepřetržitý dopředu** / **Instrukce dopředu** / `ERR_SIGSUPSEARCH` a `ERR_PERSSUPSEARCH`: Žádná pozice není vrácena a pohyb se vždy zastaví tak rychle jak je to možné na začátku dráhy hledání. Systémová proměnná `ERRNO` je nastavena na `ERR_SIGSUPSEARCH` nebo `ERR_PERSSUPSEARCH` podle použitého argumentu (signál nebo perzistentní proměnná) a chyba může být ošetřena v chybovém handleru rutiny.
- **Instrukce dozadu**: Během zpětného vykonávání provádí instrukce pohyb bez jakéhokoliv dohledu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.218 SearchL - Hledá lineárně pomocí robotu

RobotWare - OS

Pokračování

Příklad

```
VAR num fk;
...
MoveL p10, v100, fine, tool1;
SearchL \Stop, di1, sp, p20, v100, tool1;
...
ERROR
  IF ERRNO=ERR_WHLSEARCH THEN
    StorePath;
    MoveL p10, v100, fine, tool1;
    RestoPath;
    ClearPath;
    StartMove;
    RETRY;
  ELSEIF ERRNO=ERR_SIGSUPSEARCH THEN
    TPWrite "The signal of the SearchL instruction is already
         high!";
    TPReadFK fk,"Try again after manual reset of signal ?","YES",
         stEmpty, stEmpty, stEmpty, "NO";
    IF fk = 1 THEN
      StorePath;
      MoveL p10, v100, fine, tool1;
      RestoPath;
      ClearPath;
      StartMove;
      RETRY;
    ELSE
      Stop;
    ENDIF
  ENDIF
```

Jestliže signál je již aktivní na začátku procesu hledání nebo komunikace se signálem je ztracena, bude aktivován uživatelský dialog (`TPReadFK ...;`). Resetujte signál a stiskněte YES na uživatelském dialogu a robot se posune zpět k `p10` a provede další pokus. Jinak bude vykonávání programu zastaveno.

Jestliže signál je pasivní na začátku procesu hledání, robot bude hledat od pozice `p10` k `p20`. Jestliže se neobjeví žádné zjištění signálu, robot se posune zpět k `p10` a provede další pokus.

Syntaxe

```
SearchL
  [ '\' Stop ','] | [ '\' PStop ','] | [ '\' SStop ','] | [ '\'
       Sup ','']
  [ Signal ':=' ] < variable (VAR) of signaldi > |
  [ PersBool ':=' ] < persistent (PERS) of bool >
  ['\' Flanks] |
  ['\' PosFlank] |
  ['\' NegFlank] |
  ['\' HighLevel] |
  ['\' LowLevel] ','
  [ SearchPoint ':=' ] < var or pers (INOUT) of robtarget > ','
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
[ ToPoint ':=' ] < expression (IN) of robtarget >
[ '\' ID ':=' < expression (IN) of identno >] ','
[ Speed ':=' ] < expression (IN) of speeddata >
[ '\' V ':=' < expression (IN) of num > ] |
[ '\' T ':=' < expression (IN) of num > ] ','
[ Tool ':=' ] < persistent (PERS) of tooldata >
[ '\' WObj ':=' < persistent (PERS) of wobjdata > ]
[ '\' Corr ]
[ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Kruhová hledání	*SearchC - Hledá kruhově pomocí robotu na str 585*
Zapisuje do vstupu korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
Pohybuje robotem lineárně	*MoveL - Pohybuje robotem lineárně na str 411*
Lineární pohyb	*Technická referenční příručka - Přehled RAPID*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Používání chybových handlerů	*Technická referenční příručka - Přehled RAPID*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.219 SenDevice - Připojit k zařízení senzoru

Použití

`SenDevice` se používá pro připojení k zařízení senzoru připojenému k sériovému rozhraní senzoru.

Rozhraní senzoru komunikuje se senzory přes sériové kanály pomocí transportního protokolu RTP1.

Toto je příklad konfigurace kanálu senzoru.

COM_PHY_CHANNEL:

- Name “COM1:”
- Connector “COM1”
- Baudrate 19200

COM_TRP:

- Name “sen1:”
- Type “RTP1”
- PhyChannel “COM1”

Základní příklady

Následující příklad názorně ukazuje instrukci `SenDevice`:

Příklad 1

```
! Define variable numbers
CONST num SensorOn := 6;
CONST num XCoord := 8;
CONST num YCoord := 9;
CONST num ZCoord := 10;
VAR pos SensorPos;
! Connect to the sensor device“ sen1:” (defined in sio.cfg).
SenDevice "sen1:";
! Request start of sensor meassurements
WriteVar "sen1:", SensorOn, 1;
! Read a cartesian position from the sensor.
SensorPos.x := ReadVar "sen1:", XCoord;
SensorPos.y := ReadVar "sen1:", YCoord;
SensorPos.z := ReadVar "sen1:", ZCoord;
! Stop sensor
WriteVar "sen1:", SensorOn, 0;
```

Argumenty

```
SenDevice device
```

`device`

Datový typ: `string`

Jméno I/O zařízení konfigurované v sio.cfg pro použitý senzor.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
ReadBlock
  [ device‘ :=’ ] < expression(IN) of string>’,’
  [ BlockNo’ :=’ ] < expression (IN) of num > ‘,’
  [ FileName’ :=’ ] < expression (IN) of string > ‘;’
```

Související informace

Pro informace o	Viz
Zapsat proměnnou senzoru	*WriteVar - Zapsat proměnnou na str 998*
Přečíst proměnnou senzoru	*ReadVar - Přečíst proměnnou ze zařízení na str 1297*
Zapsat datový blok senzoru	*WriteBlock - Zapsat blok dat do zařízení na str 988*
Konfigurace komunikace senzoru	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.220 Set - Nastavuje digitální výstupní signál

Použití

`Set` se používá pro nastavení hodnoty digitálního výstupního signálu na jedna.

Základní příklady

Následující příklady názorně ukazují instrukci `Set`:

Příklad 1

```
Set do15;
```

Signál `do15` je nastaven na 1.

Příklad 2

```
Set weldon;
```

Signál `weldon` je nastaven na 1.

Argumenty

```
Set Signal
```

`Signal`

Datový typ: `signaldo`

Jméno signálu, který bude nastaven na jedna.

Vykonávání programu

Krátká prodleva se projevuje předtím, než signál fyzicky získá svoji novou hodnotu. Jestliže nechcete, aby vykonávání programu pokračovalo, dokud signál nezíská svoji novou hodnotu, potom můžete použít instrukci `SetDO` s volitelným parametrem `\Sync`.

Absolutní (True) hodnota závisí na konfiguraci signálu. Jestliže je signál invertován v systémových parametrech, potom tato instrukce zajistí nastavení fyzického kanálu na nulu.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
Set
  [ Signal ':=' ] < variable (VAR) of signaldo > ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Nastavení digitálního výstupního signálu na nulu	*Reset - Resetuje digitální výstupní signál na str 542*
Změnit hodnotu digitálního výstupního signálu	*SetDO - Mění hodnotu digitálního výstupního signálu na str 629*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.221 SetAllDataVal - Nastavit hodnotu všech datovým objektům v definované sadě

Použití

`SetAllDataVal`(***Set All Data Value***) umožňuje nastavit novou hodnotu všem datovým objektům určitého typu, který odpovídá dané gramatice.

Základní příklady

Následující příklad názorně ukazuje instrukci `SetAllDataVal`:

```
VAR mydata mydata0:=0;
...
SetAllDataVal "mydata"\TypeMod:="mytypes"\Hidden,mydata0;
```

Toto nastaví všechny datové objekty datového typu `mydata` v systému na stejnou hodnotu, kterou má proměnná `mydata0` (v příkladu na `0`). Uživatelsky definovaný datový typ `mydata` je definován v modulu `mytypes`.

Argumenty

```
SetAllDataVal Type [\TypeMod] [\Object] [\Hidden] Value
```

`Type`

Datový typ: `string`

Typové jméno datových objektů, které budou nastaveny.

`[ \TypeMod ]`

Typový modul

Datový typ: `string`

Jméno modulu, kde je datový typ definován, jestliže používáme uživatelem definované datové typy.

`[ \Object ]`

Datový typ: `string`

Výchozím chováním je nastavit všechny datové objekty datového typu uvedené shora, ale tento doplněk umožňuje pojmenovat jeden nebo několik objektů regulérním výrazem. (viz také instrukce `SetDataSearch`)

`[ \Hidden ]`

Datový typ: `switch`

Toto také souhlasí s datovými objekty, které jsou v rutinách (data nebo parametry rutin) skryty některými rutinami v řetězci volání.

`Value`

Datový typ: `anytype`

Proměnná, která drží novou hodnotu, která bude nastavena. Datový typ musí být stejný jako datový typ pro objekt, který bude nastaven.

Vykonávání programu

Instrukce selže, jestliže specifikace pro `Type` nebo `TypeMod` je nesprávná.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže odpovídající datový objekt je pole, potom všechny prvky pole budou nastaveny na určenou hodnotu.

Jestliže odpovídající datový objekt jsou data pouze pro čtení, potom nebude hodnota změněna.

Jestliže systém nemá žádné odpovídající datové objekty, potom to instrukce akceptuje a úspěšně vrátí.

Omezení

U polohodnotového datového typu není možné hledat datový typ s propojenou hodnotou. Například, při hledání `dionum`, nedostanete žádný nález pro signál `signaldi`, a jestliže budete hledat `num`, nedostanete žádný nález pro signály `signalgi` ani `signalai`.

Není možné nastavit hodnotu proměnné deklarované jako `LOCAL` ve vestavěném modulu RAPID.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_SYMBOL_TYPE`	Datový objekt a proměnná použité v argumentu `Value` jsou odlišného typu. Při používání datových typů `ALIAS` dostanete také tuto CHYBU, i když typy mohou mít stejný základní datový typ.

Syntaxe

```
SetAllDataVal
  [ Type ':=' ] < expression (IN) of string >
  ['\'TypeMod' :='<expression (IN) of string>]
  ['\'Object' :='<expression (IN) of string>]
  ['\'Hidden ] ','
  [ Value ':='] <variable (VAR) of anytype>';'
```

Související informace

Pro informace o	Viz
Definovat sadu symbolů ve vyhledávací akci	*SetDataSearch - Definovat sadu symbolů ve vyhledávací sekvenci na str 622*
Získat další odpovídající symbol	*GetNextSym - Získat další odpovídající symbol na str 1175*
Získat hodnotu datového objektu.	*GetDataVal - Získat hodnotu datového objektu na str 224*
Nastavit hodnotu datového objektu	*SetDataVal - Nastavit hodnotu datového objektu na str 626*
Související datový typ datapos	*datapos - Přiložený blok pro datový objekt na str 1482*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.222 SetAO - Mění hodnotu analogového výstupního signálu

Použití

`SetAO` se používá pro změnu hodnoty analogového výstupního signálu.

Základní příklady

Následující příklad názorně ukazuje instrukci `SetAO`:

Viz také *Další příklady na str 621*.

Příklad 1

```
SetAO ao2, 5.5;
```

Signál `ao2` je nastaven na `5.5`.

Argumenty

```
SetAO Signal Value
```

`Signal`

Datový typ: `signalao`

Jméno analogového výstupního signálu, který bude změněn.

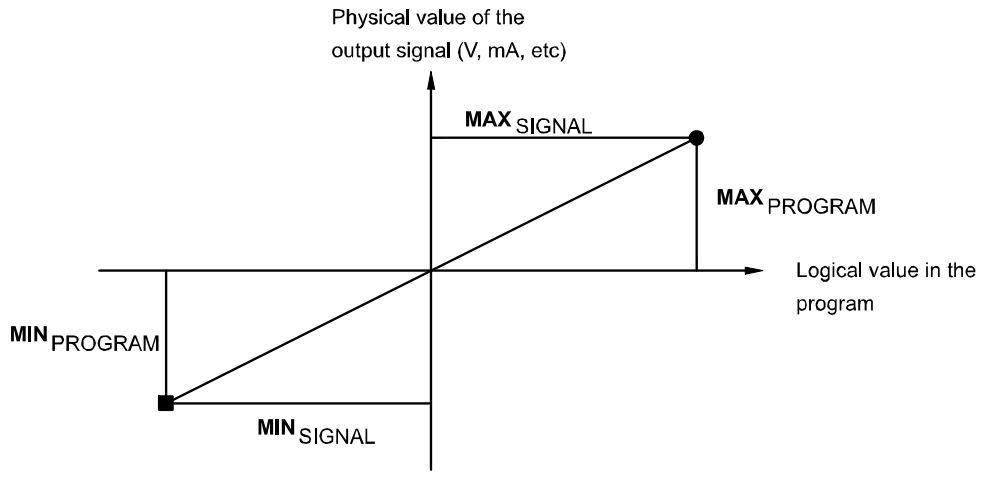
`Value`

Datový typ: `num`

Požadovaná hodnota signálu.

Vykonávání programu

Naprogramovaná hodnota je škálována (v souladu se systémovými parametry) předtím, než je odeslána na fyzický kanál. Na obrázku dole je schéma, jak jsou škálovány hodnoty analogových signálů.



xx0500002408

Řešení chyb

Následující odstranitelná chyba může být generována. Chybu je možné řešit v chybovém handleru. Systémová proměnná `ERRNO` bude nastavena na:

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

ERR_AO_LIM

jestliže naprogramovaný argument Value pro určený analogový výstupní signál Signal je mimo limity.

ERR_NO_ALIASIO_DEF jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.

ERR_NORUNUNIT

jestliže není žádný kontakt s jednotkou I/O.

ERR_SIG_NOT_VALID jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Další příklady

Více příkladů instrukce SetAO je názorně uvedeno dole.

Příklad 1

```
SetAO weldcurr, curr_outp;
```

Signál weldcurr se nastaví na stejnou hodnotu, jako je aktuální hodnota proměnné curr_outp.

Syntaxe

```
SetAO
  [ Signal ':=' ] < variable (VAR) of signalao > ','
  [ Value ':=' ] < expression (IN) of num > ';'
```

Související informace

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.223 SetDataSearch - Definovat sadu symbolů ve vyhledávací sekvenci

Použití

`SetDataSearch` se používá společně s funkcí `GetNextSym` k získání datových objektů ze systému.

Základní příklady

Následující příklad názorně ukazuje instrukci `SetDataSearch`:

Příklad 1

```
VAR datapos block;
VAR string name;
...
SetDataSearch "robtarget"\InTask;
WHILE GetNextSym(name,block \Recursive) DO
...
```

Tato akce najde všechny objekty `robtarget` v úloze.

Argumenty

```
SetDataSearch Type [\TypeMod] [\Object] [\PersSym]
     [\VarSym][\ConstSym] [\InTask] | [\InMod]
     [\InRout][\GlobalSym] | [\LocalSym]
```

`Type`

Datový typ: `string`

Jméno datového typu datových objektů, které budou vyhledány.

`[ \TypeMod ]`

Type Module

Datový typ: `string`

Jméno modulu, kde je datový typ definován, jestliže používáme uživatelem definované datové typy.

`[ \Object ]`

Datový typ: `string`

Výchozím chováním je nastavit všechny datové objekty datového typu uvedené shora, ale tento doplněk umožňuje pojmenovat jeden nebo několik objektů regulérním výrazem.

Regulérní výraz je výkonný mechanismus pro specifikaci gramatiky kvůli shodě jmen datových objektů. Řetězec se může skládat buď z běžných znaků nebo meta znaků. Meta znak je speciální operátor používaný k zastoupení jednoho nebo více běžných znaků v řetězci za účelem rozšířit hledání. Je možné pozorovat, jestli řetězec souhlasí s určeným vzorem jako celek nebo hledat v řetězci podřetězec, který souhlasí s určeným vzorem.

V regulérním výrazu jsou všechny alfanumerické znaky vzájemně ve shodě. Je možné říci, že vzorec „abc“ bude souhlasit pouze s datovým objektem pojmenovaným „abc“. Aby souhlasila všechna jména datových objektů obsahujících

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

znakovou sekvenci „abc“, je nezbytné přidat některé meta znaky. Regulérním výrazem pro to je ".*abc.*".

Dostupná sada meta znaků je uvedena dole.

Výraz	Význam
.	Jakýkoliv jednoduchý znak.
[s]	Jakýkoliv jednoduchý znak v neprázdné sadě s, kde s je sekvence znaků. Rozsahy mohou být určeny jako c-c.
[^s]	Jakýkoliv jednoduchý znak, který není v sadě s.
r*	Žádný nebo více výskytů regulérního výrazu r.
r+	Jeden nebo více výskytů regulérního výrazu r.
r?	Žádný nebo jeden výskyt regulérního výrazu r.
(r)	Regulérní výraz r. Používá se pro oddělení regulérního výrazu od jiného.
r | r’	Regulérní výrazy r nebo r’.
.*	Jakákoliv znaková sekvence (žádný, jeden nebo několik znaků).

Výchozím chováním je akceptovat všechny symboly, ale jestliže je určen jeden nebo několik následujících `PersSym`, `VarSym`, nebo `ConstSym`, potom jsou akceptovány pouze symboly, které odpovídají specifikaci:

`[ \PersSym ]`

Persistent Symbols

Datový typ: `switch`

Akceptovat symboly perzistentní proměnné (`PERS`).

`[ \VarSym ]`

Variable Symbols

Datový typ: `switch`

Akceptovat symboly proměnné (`VAR`).

`[ \ConstSym ]`

Constant Symbols

Datový typ: `switch`

Akceptovat symboly konstanty(`CONST`).

Jestliže není určen ani jeden příznak `\InTask` nebo `\InMod` , potom je hledání spuštěno na systémové úrovni. Systémová úroveň je kořen ke všem jiným definicím symbolů ve stromu symbolů. Na systémové úrovni jsou všechny vestavěné symboly umístěny plus odkaz k úrovni úlohy. Na úrovni úlohy jsou všechny načtené globální symboly umístěny plus odkaz k úrovni modulů.

Jestliže je nastaven příznak `\Recursive` v `GetNextSym`, potom vyhledávací akce vstoupí do všech načtených modulů a rutin pod systémovou úrovní.

`[ \InTask ]`

In Task

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.223 SetDataSearch - Definovat sadu symbolů ve vyhledávací sekvenci
RobotWare - OS
Pokračování

Začněte s hledáním na úrovni úlohy. Na úrovni úlohy jsou všechny načtené globální symboly umístěny plus odkaz k úrovni modulů.

Jestliže je nastaven příznak `\Recursive` v `GetNextSym`, potom vyhledávací akce vstoupí do všech načtených modulů a rutin pod úrovní úlohy.

`[ \InMod ]`

In Module

Datový typ: `string`

Začněte s hledáním na úrovni určeného modulu. Na úrovni modulu jsou všechny načtené globální a lokální symboly deklarované v určeném modulu umístěny plus odkaz k úrovni rutin.

Jestliže je nastaven příznak `\Recursive` v `GetNextSym`, potom vyhledávací akce vstoupí do všech načtených rutin pod úrovní určeného modulu (deklarováno v určeném modulu).

`[ \InRout ]`

In Routine

Datový typ: `string`

Hledat pouze na úrovni určené rutiny.

Jméno modulu pro rutinu musí být určeno v argumentu `\InMod`.

Výchozím chováním je být ve shodě s lokálními a globálními symboly modulů, ale jestliže je určen jeden z následujících `\GlobalSym`, nebo `\LocalSym`, potom jsou akceptovány pouze symboly, které odpovídají specifikaci:

`[ \GlobalSym ]`

Global Symbols

Datový typ: `switch`

Přeskočit symboly lokálních modulů.

`[ \LocalSym ]`

Local Symbols

Datový typ: `switch`

Přeskočit symboly globálních modulů.

Vykonávání programu

Instrukce selže, jestliže specifikace pro jeden z `Type`, `TypeMod`, `InMod` nebo `InRout` je nesprávná.

Jestliže systém nemá žádné odpovídající objekty, potom to instrukce akceptuje a úspěšně vrátí, ale první `GetNextSym` vrátí `FALSE`.

Omezení

Datové objekty pole nemohou být definovány v sadě hledání symbolů a nemohou být nalezeny ve vyhledávací sekvenci.

U polohodnotového datového typu není možné hledat datový typ s propojenou hodnotou. Například, při hledání `dionum`, nedostanete žádný nález pro signál

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

signaldi, a jestliže budete hledat num, nedostanete žádný nález pro signály signalgi ani signalai.

Instalované vestavěné symboly deklarované jako LOCAL nebudou nikdy nalezeny, bez ohledu na použití argumentu \GlobalSym, \LocalSym nebo žádného z nich.

Instalované vestavěné symboly deklarované jako globální nebo jako TASK budou vždy nalezeny, bez ohledu na použití argumentu \GlobalSym, \LocalSym nebo žádného z nich.

Není možné používat SetDataSearch pro hledání dat některých datových typů ALIAS definovaných s kódem RAPID. U předdefinovaného datového typu ALIAS není žádné omezení.

Syntaxe

```
SetDataSearch
  [ Type ':=' ] < expression (IN) of string >
  ['\'TypeMod ':='<expression (IN) of string>]
  ['\'Object ':='<expression (IN) of string>]
  ['\'PersSym ]
  ['\'VarSym ]
  ['\'ConstSym ]
  ['\'InTask ]
  | ['\'InMod' :='<expression (IN) of string>]
  ['\'InRout ':='<expression (IN) of string>]
  ['\'GlobalSym ]
  | ['\'LocalSym]' ;'
```

Související informace

Pro informace o	Viz
Získat další odpovídající symbol	*GetNextSym - Získat další odpovídající symbol na str 1175*
Získat hodnotu datového objektu.	*GetDataVal - Získat hodnotu datového objektu na str 224*
Nastavit hodnotu mnoha datových objektů	*SetAllDataVal - Nastavit hodnotu všech datovým objektům v definované sadě na str 618*
Související datový typ datapos	*datapos - Přiložený blok pro datový objekt na str 1482*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.224 SetDataVal - Nastavit hodnotu datového objektu

Použití

`SetDataVal` (*Set Data Value*) umožňuje nastavit hodnotu pro datový objekt, který je určen s proměnnou řetězce.

Základní příklady

Následující příklady názorně ukazují instrukci `SetDataVal`:

Příklad 1

```
VAR num value:=3;
...
SetDataVal "reg"+ValToStr(ReadNum(mycom)),value;
```

Toto nastaví hodnotu `3` pro registr s číslem, které je přijato od sériového kanálu `mycom`.

Příklad 2

```
VAR datapos block;
VAR bool truevar:=TRUE;
...
SetDataSearch "bool" \Object:="my.*" \InMod:="mymod"\LocalSym;
WHILE GetNextSym(name,block) DO
  SetDataVal name\Block:=block,truevar;
ENDWHILE
```

Tato akce nastaví všechny lokální `bool`, které začínají s `my` v modulu `mymod` na `TRUE`.

Příklad 3

```
VAR string StringArrVar_copy{2};
...
StringArrVar_copy{1} := "test1";
StringArrVar_copy{2} := "test2";
SetDataVal "StringArrVar", StringArrVar_copy;
```

Tato akce nastaví pole `StringArrVar` tak, aby obsahovalo dva řetězce `test1` a `test2`.

Argumenty

```
SetDataVal Object [\Block]|[\TaskRef]|[\TaskName] Value
```

`Object`

Datový typ: `string`

Jméno datového objektu.

`[ \Block ]`

Datový typ: `datapos`

Blok vložený k datovému objektu. Toto může být získáno pouze s funkcí `GetNextSym`.

Jestliže je tento argument vypuštěn, bude nastavena hodnota viditelného datového objektu v rámci současného vykonávání programu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[\TaskRef]`

Task Reference

Datový typ: `taskid`

Identita programové úlohy, ve které se má hledat určený datový objekt. Při použití tohoto argumentu můžete hledat deklarace `PERS` nebo `TASKPERS` v jiných úlohách, všechny ostatní deklarace povedou k chybě.

Pro všechny programové úlohy v systému budou dostupné předdefinované proměnné datového typu `taskid`. Identita proměnné bude například "taskname"+"Id", pro úlohu `T_ROB1` bude identita proměnné `T_ROB1Id`.

`[\TaskName]`

Datový typ: `string`

Jméno programové úlohy, ve které se má hledat určený datový objekt. Při použití tohoto argumentu můžete hledat deklarace `PERS` nebo `TASKPERS` v jiných úlohách, všechny ostatní deklarace povedou k chybě.

`Value`

Datový typ: `anytype`

Proměnná, která drží novou hodnotu k nastavení. Datový typ musí být stejný jako datový typ pro datový objekt, který bude nastaven. Hodnota nastavení musí být získána z proměnné, ale může být uložena do proměnné nebo perzistentu.

Řešení chyb

Systémová proměnná `ERRNO` je nastavena na `ERR_SYM_ACCESS`, jestliže:

- datový objekt neexistuje
- datovým objektem jsou data pouze pro čtení
- datový objekt jsou data rutiny nebo parametr rutiny a není umístěn v aktuální aktivní rutině
- hledání v jiných úlohách pro jiné deklarace než `PERS` nebo `TASK PERS`

Při používání argumentů `TaskRef` nebo `TaskName` můžete hledat deklarace `PERS` nebo `TASK PERS` v jiných úlohách, všechny ostatní deklarace budou mít za výsledek chybu a systémová proměnná `ERRNO` se nastaví na `ERR_SYM_ACCESS`. Hledání `PERS` deklarovaného jako `LOCAL` v jiných úlohách bude mít za výsledek také chybu a systémová proměnná `ERRNO` se nastaví na `ERR_SYM_ACCESS`.

Systémová proměnná `ERRNO` se nastaví na `ERR_INVDIM`, jestliže datový objekt a proměnná použité v argumentu `Value` mají odlišné dimenze.

Systémová proměnná `ERRNO` se nastaví na `ERR_SYMBOL_TYPE`, jestliže datový objekt a proměnná použité v argumentu `Value` jsou různé typy. Při použití datových typů `ALIAS` dostanete také tuto CHYBU, i když typy mohou mít stejný datový typ základny.

Chyba může být zpracována v chybovém handleru rutiny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.224 SetDataVal - Nastavit hodnotu datového objektu

RobotWare - OS

Pokračování

Omezení

U polohodnotového datového typu není možné hledat datový typ s propojenou hodnotou. Například, při hledání `dionum`, nedostanete žádný nález pro signál `signaldi`, a jestliže budete hledat `num`, nedostanete žádný nález pro signály `signalgi` ani `signalai`.

Není možné nastavit hodnotu proměnné deklarované jako `LOCAL` ve vestavěném modulu RAPID.

Syntaxe

```
SetDataVal
  [ Object ':=' ] < expression (IN) of string >
  ['\'Block' :='<variable (VAR) of datapos>]
  |[ '\'TaskRef' :=' <variable (VAR) of taskid>]
  |[ '\'TaskName' :=' <expression (IN) of string>] ',']
  [ Value ':=' ] <variable (VAR) of anytype>]';'
```

Související informace

Pro informace o	Viz
Definovat sadu symbolů ve vyhledávací akci	*SetDataSearch - Definovat sadu symbolů ve vyhledávací sekvenci na str 622*
Získat další odpovídající symbol	*GetNextSym - Získat další odpovídající symbol na str 1175*
Získat hodnotu datového objektu.	*GetDataVal - Získat hodnotu datového objektu na str 224*
Nastavit hodnotu mnoha datových objektů	*SetAllDataVal - Nastavit hodnotu všech datovým objektům v definované sadě na str 618*
Související datový typ `datapos`	*datapos - Přiložený blok pro datový objekt na str 1482*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.225 SetDO - Mění hodnotu digitálního výstupního signálu

Použití

`SetDO` se používá pro změnu hodnoty digitálního výstupního signálu s nebo bez časové prodlevy nebo synchronizace.

Základní příklady

Následující příklady názorně ukazují instrukci `SetDO`:

Příklad 1

```
SetDO do15, 1;
```

Signál `do15` je nastaven na `1`.

Příklad 2

```
SetDO weld, off;
```

Signál `weld` je nastaven na `off`.

Příklad 3

```
SetDO \SDelay := 0.2, weld, high;
```

Signál `weld` je nastaven na `high` s prodlevou `0.2` s. Vykonávání programu pokračuje s další instrukcí.

Příklad 4

```
SetDO \Sync ,do1, 0;
```

Signál `do1` je nastaven na `0`. Vykonávání programu čeká, dokud signál nebude fyzicky nastaven na určenou hodnotu.

Argumenty

```
SetDO [ \SDelay ]|[ \Sync ] Signal Value
```

`[ \SDelay ]`

Signal Delay

Datový typ: `num`

Odkládá změnu o určitou dobu danou v sekundách (max. 2000 s). Vykonávání programu pokračuje přímo s další instrukcí. Po dané časové prodlevě je signál změněn, bez toho, že by bylo ovlivněno vykonávání zbytku programu.

`[ \Sync ]`

Synchronization

Datový typ: `switch`

Jestliže se použije tento argument, potom bude vykonávání programu čekat, dokud signál nebude fyzicky nastaven na určenou hodnotu.

`Signal`

Datový typ: `signaldo`

Jméno signálu, který bude změněn.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.225 SetDO - Mění hodnotu digitálního výstupního signálu

RobotWare - OS
Pokračování

`Value`

Datový typ: `dionum`

Požadovaná hodnota signálu 0 nebo 1.

Určená hodnota	Nastavit digitální výstup na
0	0
Každá hodnota kromě 0	1

Vykonávání programu

Absolutní (true) hodnota závisí na konfiguraci signálu. Jestliže je signál invertován v systémových parametrech, hodnota fyzického kanálu je opačná.

Jestliže je použit některý z argumentů `\SDelay` nebo `\Sync`, potom bude signál nastaven tak rychle, jak je to možné, a další instrukce bude provedena okamžitě, bez čekání na fyzické nastavení signálu.

Omezení

Jestliže `SetDO` s argumentem `\SDelay` je následován novým `SetDO` na stejném signálu s nebo bez argumentu `\SDelay`, potom bude první `SetDO` zrušen, jestliže druhý `SetDO` je vykonán ještě předtím, než doba prodlevy prvního `SetDO` vypršela.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_ARGVALERR`, **jestliže hodnota pro argument** `SDelay` **překračuje max přípustnou hodnotu (2000 s).**

`ERR_NO_ALIASIO_DEF` **jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí** `AliasIO`.

`ERR_NORUNUNIT` **jestliže není žádný kontakt s jednotkou I/O.**

`ERR_SIG_NOT_VALID` **jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).**

Syntaxe

```
SetDO
  [ '\' SDelay ':=' < expression (IN) of num > ',' ]
  |['\'Sync',']
  [ Signal ':=' ] < variable (VAR) of signaldo > ','
  [ Value ':=' ] < expression (IN) of dionum > ';'
```

Související informace

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.226 SetGO - Mění hodnotu skupiny digitálních výstupních signálů

Použití

`SetGO` se používá pro změnu hodnoty skupiny digitálních výstupních signálů s nebo bez časové prodlevy.

Základní příklady

Následující příklady názorně ukazují instrukci `SetGO`:

Příklad 1

```
SetGO go2, 12;
```

Signál `go2` je nastaven na `12`. Jestliže `go2` obsahuje 4 signály, např. výstupy 6-9, potom výstupy 6 a 7 jsou nastaveny na nulu, zatímco výstupy 8 a 9 jsou nastaveny na jedna.

Příklad 2

```
SetGO \SDelay := 0.4, go2, 10;
```

Signál `go2` je nastaven na `10`. Jestliže `go2` obsahuje 4 signály, např. výstupy 6-9, potom výstupy 6 a 8 jsou nastaveny na nulu, zatímco výstupy 7 a 9 jsou nastaveny na jedna s prodlevou `0.4` s. Vykonávání programu pokračuje s další instrukcí.

Příklad 3

SetGO go32, 4294967295;

Signál `go32` je nastaven na `4294967295`. `go32` obsahuje 32 signálů, které jsou všechny nastaveny na jedna.

Argumenty

```
SetGO [ \SDelay ] Signal Value | Dvalue
```

`[ \SDelay ]`

Signal Delay

Datový typ: `num`

Odkládá změnu o určitou dobu danou v sekundách (max. 2000 s). Vykonávání programu pokračuje přímo s další instrukcí. Po dané časové prodlevě je hodnota signálů změněna, bez toho, že by bylo ovlivněno vykonávání zbytku programu.

Jestliže argument je vypuštěn, potom jsou hodnoty signálů změněny přímo.

`Signal`

Datový typ: `signalgo`

Jméno skupiny signálů, které bude změněno.

`Value`

Datový typ: `num`

Požadovaná hodnota skupiny signálů (kladné celé číslo) je uvedena v tabulce dole.

Přípustná hodnota závisí na počtu signálů ve skupině. Datový typ `num` může držet hodnotu pro skupinu 23 signálů nebo méně.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.226 SetGO - Mění hodnotu skupiny digitálních výstupních signálů
RobotWare - OS
Pokračování

`Dvalue`

Datový typ: `dnum`

Požadovaná hodnota skupiny signálů (kladné celé číslo) je uvedena v tabulce dole.

Přípustná hodnota závisí na počtu signálů ve skupině. Datový typ dnum může držet hodnotu pro skupinu 32 signálů nebo méně.

Počet signálů	Přípustná hodnota	Přípustná Dhodnota
1	0-1	0-1
2	0-3	0-3
3	0-7	0-7
4	0-15	0-15
5	0-31	0-31
6	0-63	0-63
7	0-127	0-127
8	0-255	0-255
9	0-511	0-511
10	0-1023	0-1023
11	0-2047	0-2047
12	0-4095	0-4095
13	0-8191	0-8191
14	0-16383	0-16383
15	0-32767	0-32767
16	0-65535	0-65535
17	0-131071	0-131071
18	0-262143	0-262143
19	0-524287	0-524287
20	0-1048575	0-1048575
21	0-2097151	0-2097151
22	0-4194303	0-4194303
23	0-8388607	0-8388607
24	*	0-16777215
25	*	0-33554431
26	*	0-67108863
27	*	0-134217727
28	*	0-268435455
29	*	0-536870911
30	*	0-1073741823
31	*	0-2147483647
32	*	0-4294967295

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

*) Argument `Value` typu `num` může držet pouze max 23 signálů ve srovnání s argumentem `Dvalue` typu `dnum`, který může držet až 32 signálů.

Vykonávání programu

Naprogramovaná hodnota je převedena na neznaménkové binární číslo. Toto binární číslo je odesláno na skupinu signálů s výsledkem, že individuální signály ve skupině jsou nastaveny na 0 nebo 1. Kvůli interním prodlevám může být hodnota signálu nedefinována na krátký časový úsek.

Omezení

Max počet signálů, které se mohou použít pro skupinu, je 23, jestliže je použit argument `Value`, a 32, jestliže je použit argument `Dvalue`. Toto omezení je platné pro všechny instrukce a funkce využívající skupinové signály.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_ARGVALERR`, jestliže hodnota pro argument `SDelay` překračuje max přípustnou hodnotu (2000 s).

`ERR_GO_LIM`, jestliže naprogramovaný argument `Value` nebo `Dvalue` pro určený digitální skupinový výstupní signál `Signal` je mimo limity.

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
SetGO
  [ '\' SDelay ':=' < expression (IN) of num > ',' ]
  [ Signal ':=' ] < variable (VAR) of signalgo > ','
  [ Value ':=' ] < expression (IN) of num >
  | [ Dvalue' :=' ] < expression (IN) of dnum > ';'
```

Související informace

Pro informace o	Viz
Ostatní instrukce vstup/výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O (systémové parametry)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.227 SetSysData - Nastavit systémová data

Použití

`SetSysData` aktivuje určené jméno systémových dat pro určený datový typ.

S touto instrukcí je možné změnit aktuální aktivní nástroj, pracovní objekt, užitečnou zátěž nebo celkovou zátěž pro robot v aktuální nebo připojené pohybové úloze.

Základní příklady

Následující příklad názorně ukazuje instrukci `SetSysData`:

Příklad 1

```
SetSysData tool5;
```

Nástroj `tool5` je aktivován.

```
SetSysData tool0 \ObjectName := "tool6";
```

Nástroj `tool6` je aktivován.

```
SetSysData anytool \ObjectName := "tool2";
```

Nástroj `tool2` je aktivován.

Argumenty

```
SetSysData SourceObject [\ObjectName]
```

`SourceObject`

Datový typ:`anytype`

Perzistentní proměnná, která by měla být aktivní jako aktuální systémová data.

Datový typ tohoto argumentu také určuje typ systémových dat, která budou aktivována pro robot v aktuální nebo připojené pohybové úloze.

Datový typ	Typ systémových dat
tooldata	Nástroj
wobjdata	Pracovní objekt
loaddata	Užitečná zátěž/Celková zátěž

Celé pole nebo komponent záznamu se nemohou používat.

`[ \ObjectName ]`

Datový typ:`string`

Jestliže je určen tento volitelný argument, potom to určuje jméno datového objektu, který bude aktivní (potlačuje jméno určené v argumentu `SourceObject`). Datový typ datového objektu, který bude aktivní, je vždy získáno z argumentu `SourceObject`.

Vykonávání programu

Aktuálně aktivní systémový datový objekt pro nástroj, pracovní objekt, užitečnou zátěž nebo celkovou zátěž se nastavuje podle argumentů.

Všimněte si, že tato instrukce pouze aktivuje nový datový objekt (nebo stejný jako předtím) a nikdy nemění hodnotu žádného datového objektu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
SetSysData
[ SourceObject':='] < persistent(PERS) of anytype>
['\'ObjectName':=' < expression (IN) of string> ] ';'
```

Související informace

Pro informace o	Viz
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice užitečné zátěže	*loaddata - Zátěžová data na str 1523*
Získat systémová data	*GetSysData - Získat systémová data na str 227*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.228 SetupCyclicBool - Nastavit cyklicky hodnocenou logickou podmínku

Použití

`SetupCyclicBool` se používá pro nastavení logické podmínky, která bude cyklicky hodnocena a přidělena k perzistentní booleánské proměnné.

Základní příklady

Následující příklad názorně ukazuje instrukci `SetupCyclicBool`.

Viz také *Další příklady na str 637*.

Příklad 1

```
PERS bool cyclicflag1;

PROC main()
  SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
  ...
```

Nastavuje cyklické hodnocení logické podmínky `di1=1 AND do2=1` a přiděluje výsledek k perzistentní booleánské proměnné `cyclicflag1`.

Argumenty

```
SetupCyclicBool Flag Cond
```

`Flag`

Datový typ: `bool`

Booleovská proměnná perzistentu, která obsahuje uloženou hodnotu logické podmínky.

Proměnná musí být deklarována jako

`Cond`

Datový typ: `bool`

Logický výraz, který by měl být hodnocen cyklicky.

Výraz se může skládat z:

- Konstanty nebo perzistentní proměnné typů `bool`, `num` a `dnum` (a alias od `bool`, `num` a `dnum`).
- Globální digitální vstupní a výstupní signály.
- Operandy: '`NOT`', '`AND`', '`OR`', '`XOR`', '`=`', '`(`', '`)`'

Vykonávání programu

S touto instrukcí je možné nastavovat komplexnější podmínky a používat cyklický příznak namísto poznání, jestli podmínka je splněna nebo nikoliv.

Cyklické hodnocení logické podmínky a přidělování k perzistentní booleánské proměnné se provádí každých 12 ms.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Výraz musí být vyhodnocen na booleánskou hodnotu TRUE nebo FALSE. Všechny části výrazu musí být také vyhodnoceny na booleánskou hodnotu TRUE nebo FALSE.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_NO_ALIASIO_DEF	Proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.
ERR_NORUNUNIT	Není žádný kontakt s jednotkou I/O.
ERR_SIG_NOT_VALID	Není přístup k I/O signálu (platí pouze pro aplikační sběrnici ICI).

Další příklady

Více příkladů instrukce SetupCyclicBool je názorně uvedeno dole.

Příklad 1

```
ALIAS bool aliasBool;
PERS bool cyclicflag1;
TASK PERS aliasBool cyclicflag2:=FALSE;
PERS aliasBool flag1:=FALSE;
TASK PERS aliasBool flag2:=FALSE;
CONST num HIGH:=1;
CONST num LOW:=0;

PROC main()
  SetupCyclicBool cyclicflag1, (di1=HIGH AND di2=HIGH AND di3=LOW)
       OR flag1=TRUE;
  SetupCyclicBool cyclicflag2, di4=HIGH AND flag2=TRUE;
  ...
  WaitUntil cyclicflag1=TRUE;
  IF cyclicflag2 = TRUE THEN
    MoveL p1, v1000, z30, tool2;
  ELSE
    MoveL p2, v1000, z30, tool2;
  ENDIF
  ...
```

Příklad nahoře nastavuje cyklické hodnocení 2 výrazů. Vykonávání čeká na nastavení cyclicflag1. cyclicflag2 rozhoduje, na kterou pozici se robot posune.

Příklad 2

```
!This condition is wrong:
SetupCyclicBool m1, 5;

!This condition is correct:
SetupCyclicBool m1, myNum = 5;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.228 SetupCyclicBool - Nastavit cyklicky hodnocenou logickou podmínku
Pokračování

První podmínka není správná, jelikož hodnota 5 není booleánská. Druhá podmínka je správná, jelikož srovnání může být vyhodnoceno jako booleánská podmínka, tj. TRUE nebo FALSE.

Syntaxe

```
SetupCyclicBool
  [ Flag ':=' ] <persistent (PERS) of bool> ','
  [ Cond ':=' ] <expression (IN) of bool> ';'
```

Související informace

Pro informace o	Viz
Odstranit cyklicky hodnocenou logickou podmínku	*RemoveCyclicBool - Odstranit cyklicky hodnocenou logickou podmínku na str 534*
Odstranit všechny cyklicky hodnocené logické podmínky	*RemoveAllCyclicBool - Odstranit všechny cyklicky hodnocené logické podmínky na str 533*
Cyklicky hodnocené logické podmínky, *Cyclic bool*.	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.229 SetupSuperv - Nastavit podmínky pro dohled signálu v CAP

Použití

`SetupSuperv` se používá pro nastavení podmínek pro I/O signály, které budou dohlíženy. Podmínky jsou shromážděny v různých seznamech:

- PRE
- PRE_START
- END_PRE
- START
- MAIN
- END_MAIN
- START_POST1
- POST1
- END_POST1
- START_POST2
- POST2
- END_POST2

Více informací o seznamech dohledů najdete v *Application manual - Continuous Application Platform*.

Jako volitelný parametr může být určen výstupní (out) signál. Tento výstupní signál je nastaven příliš vysoko, jestliže daná podmínka selže.

Základní příklad

```
PROC main()
  InitSuperv;
  SetupSuperv diWR_EST, ACT, SUPERV_MAIN \ErrIndSig:= do_WR_Sup;
  SetupSuperv diGA_EST, ACT, SUPERV_MAIN;
  CapL p2, v100, cdata1, weavestart, weave, fine, tWeldGun;
ENDPROC
```

`SetupSuperv` se používá pro nastavení dohledu nad signály. Jestliže signál *diWR_EST* selže během fáze SUPERV_MAIN, digitální výstupní signál *do_WR_Sup* je nastaven vysoko.

Instrukce `SetupSuperv` by měla být vykonávána pouze když jsou změněna data dohledu. Jestliže data dohledu nejsou nikdy měněna, je dobrým řešením umístit je do modulu, který je vykonáván ze spouštěcího zásobníku.

Argumenty

```
SetupSupervSignal Condition Listtype [\ErrIndSig]
```

Signal

Datový typ: `signaldi`

Digitální signál, který bude pod dohledem.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.229 SetupSuperv - Nastavit podmínky pro dohled signálu v CAP
Continuous Application Platform (CAP)
Pokračování

Condition

Datový typ: num

Jméno představující jednu z následujících dostupných podmínek:

ACT:	Používá se pro dohled nad statutem. Očekávaný status signálu během dohledu: aktivní. Jestliže se signál stane pasivním, dohled se spustí.
PAS:	Používá se pro dohled nad statutem. Očekávaný status signálu během dohledu: pasivní. Jestliže se signál stane aktivním, dohled se spustí.
POS_EDGE:	Používá se pro adresování dohledu. Očekávaný status signálu na konci dohledu: aktivní. Jestliže se signál nestane aktivním ve zvoleném časovém úseku, spustí se dohled.
NEG_EDGE:	Používá se pro dohled nad navazováním spojení. Očekávaný status signálu na konci dohledu: aktivní. Jestliže se signál nestane aktivním ve zvoleném časovém úseku, spustí se dohled.

Listtype

Datový typ: num

Jméno reprezentující počet různých seznamů (například fáze v procesu):

- SUPERV_PRE
- SUPERV_PRE_START
- SUPERV_END_PRE
- SUPERV_START
- SUPERV_MAIN
- SUPERV_END_MAIN
- SUPERV_START_POST1
- SUPERV_POST1
- SUPERV_END_POST1
- SUPERV_START_POST2
- SUPERV_POST2
- SUPERV_END_POST2

[\ErrIndSig]

Datový typ: signaldo

Používá se pro indikaci podmínky, která selhala, jestliže se objevila porucha. Když vznikne porucha, hodnota na tomto signálu je nastavena na 1. Jedná se o volitelný parametr.

Vykonávání programu

Daný signál a jeho podmínka jsou přidány do zvoleného seznamu. Jestliže signál selže, instrukce CapL/CapC bude hlásit, že se objevila chyba dohledu během určené fáze a který signál (-y) selhal.

Chyby

CAP_SPV_LIM

Max počet nastavených dohledů byl překročen.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

CAP_SPV_UNK_LST

Seznam dohledů je neznámý.

Omezení

Pouze digitální vstupní signály mohou být pod dohledem.

Dohled nad statutem se vztahuje na kompletní sekvenci CAP instrukcí (viz sekce *Dohled a procesní fáze* v *Application manual - Continuous Application Platform*).

Syntaxe

```
SetupSuperv
  [Signal ':='] < variable (VAR) of signaldi > ','
  [Condition ':='] < variable (IN) of num > ','
  [Listtype ':='] < variable (IN) of num >
  [\ErrIndSig ':=' < variable (VAR) of signaldo >] ';'
```

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instrukce `InitSuperv`	*InitSuperv - Resetovat veškerý dohled pro CAP na str 271*
Instrukce `RemoveSuperv`	*RemoveSuperv - Odstranit podmínku pro jeden signál na str 538*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.230 SiConnect - Připojení rozhraní senzoru

Použití

`SiConnect` se používá pro sestavení spojení k externímu zařízení.

Základní příklady

Základní příklad instrukce `SiConnect` je názorně uveden dole.

Viz také *Další příklady na str 642*.

Příklad 1

```
PERS sensor AnyDevice;
...
  SiConnect AnyDevice;
```

Sestavit spojení k zařízení nazvanému `AnyDevice`.

Argumenty

```
SiConnect Sensor [\NoStop]
```

`Sensor`

Datový typ: `sensor`

Popisovač pro externí zařízení, ke kterému je třeba se připojit. Argumentem je perzistentní proměnná a její jméno musí být stejné jako jméno určené jako klient v souboru pro nastavení *Settings.xml*.

`[\NoStop]`

Datový typ: `switch`

`\NoStop` bude zabraňovat zastavení systému, když bude zjištěna komunikační chyba se senzorem. Může to být užitečné, jestliže žádné pohyby robotu nezávisí na senzoru. Když se použije `\NoStop`, pohyby v systému budou pokračovat, i když komunikace se senzorem je ztracena.

Při používání `\NoStop` je možné provádět ošetřování chyb v rutině `TRAP` s použitím `IError` nebo `IPers`.

Vykonávání programu

Načítá aktuální konfiguraci senzoru a zakládá spojení k externímu zařízení.

Senzor zůstává připojen, i když ukazatel programu je nastaven na main.

Další příklady

Více příkladů jak používat instrukci `SiConnect` je názorně uvedeno dole.

Příklad 1

```
PERS sensor AnyDevice;
PERS robdata DataOut := [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
PERS sensdata DataIn :=
     ["No",[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
VAR num SampleRate:=64;
...
! Setup Interface Procedure
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
PROC RRI_Open()
  SiConnect AnyDevice;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, SampleRate;
  SiSetCyclic AnyDevice, DataOut, SampleRate;
ENDPROC
```

Při volání rutiny `RRI_Open` je nejprve otevřeno spojení k zařízení se jménem `AnyDevice`. Potom je spuštěn cyklický přenos rychlostí `SampleRate`.

Příklad 2

```
PERS sensor AnyDevice;
...
  SiConnect AnyDevice \NoStop;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, SampleRate;
  SiSetCyclic AnyDevice, DataOut, SampleRate;
...
TRAP sensorChange
  IF AnyDevice.state = STATE_ERROR THEN
  ...
  ENDIF
ENDTRAP
```

Založit spojení k zařízení nazvanému `AnyDevice` s volitelným argumentem `\NoStop` chránícím systém před zastavením, jestliže spojení k `AnyDevice` je přerušeno. Ošetřit chybové stavy v rutině `TRAP`.

Řešení chyb

Jestliže se jako komunikační protokol používá UDP, není dána žádná záruka týkající se úspěšné operace připojení a proto není možné ošetřování chyb v okamžiku připojení.

Jestliže se jako komunikační protokol používá TCP, systémová proměnná `ERRNO` se nastaví na `ERR_COMM_INIT`, jestliže operace připojení selže. Tato chyba může být potom ošetřena v chybovém handleru.

Přepínač `\NoStop` umožňuje ošetřovat komunikační chyby zjištěné po úspěšném připojení. `\NoStop` znamená, že pohyby a vykonávání RAPID pokračují a že rutina `TRAP` může být použita k ošetřování konkrétních chyb pomocí `IError` nebo konkrétních změn stavu pomocí `IPers`.

POZNÁMKA

`IPers` a `IError` nejsou bezpečná přerušení, takže pokud je zjištěna chyba po zastavení, žádná rutina `TRAP` nebude vykonána. Možnost, jak se vyrovnat s tímto problémem, je mít `SiConnect \NoStop` v restartovacím zásobníku, aby bylo jisté, že aplikace se snaží o nové založení spojení ke klientovi.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.230 SiConnect - Připojení rozhraní senzoru

Robot Reference Interface

Pokračování

Syntaxe

```
SiConnect
  [ Sensor ':=' ] < persistent (PERS) of sensor >
  [ '\' NoStop ] ';'
```

Související informace

Pro informace o	Viz
Blízké spojení k externímu systému.	Kontrola informačních štítků
Data registru pro cyklický přenos.	Kontrola informačních štítků
Objednat přenos cyklických dat.	*SiGetCyclic - Rozhraní senzoru se zacyklilo. na str 646*
Popisovač k externímu zařízení.	Kontrola informačních štítků
Stav komunikace zařízení.	Kontrola informačních štítků
Robot Reference Interface	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.231 SiClose - Rozhraní senzoru zavřít

Použití

`SiClose` zavírá existující spojení k externímu zařízení.

Základní příklady

Základní příklad instrukce `SiClose` je názorně uveden dole.

Příklad 1

```
PERS sensor AnyDevice;
...
  SiClose AnyDevice;
```

Zavřít spojení k zařízení nazvanému `AnyDevice`.

Argumenty

```
SiClose Sensor
```

`Sensor`

Datový typ: `sensor`

Popisovač pro externí zařízení, které by mělo být zavřeno. Argumentem je perzistentní proměnná a její jméno musí být stejné jako jméno určené jako klient v souboru pro nastavení *Settings.xml*.

Vykonávání programu

Zavírá existující spojení k externímu zařízení.

Řešení chyb

Jestliže se jako komunikační protokol používá UDP, není dána žádná záruka týkající se blízké operace a proto není možné ošetřování chyb.

Jestliže se jako komunikační protokol používá TCP, systémová proměnná `ERRNO` se nastaví na `ERR_COMM_INIT`, jestliže blízká operace selže. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
SiClose
  [ Sensor ':=' ] < persistent (PERS) of sensor > ';'
```

Související informace

Pro informace o	Viz
Založit spojení k externímu systému.	Kontrola informačních štítků
Data registru pro cyklický přenos.	Kontrola informačních štítků
Objednat přenos cyklických dat.	Kontrola informačních štítků
Popisovač k externímu zařízení.	Kontrola informačních štítků
Stav komunikace zařízení.	Kontrola informačních štítků
Robot Reference Interface	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.232 SiGetCyclic - Rozhraní senzoru se zacyklilo.

Použití

`SiGetCyclic` objednává data pro cyklický přenos z externího zařízení.

Základní příklady

Základní příklad instrukce `SiGetCyclic` je názorně uveden dole.

Viz také *Další příklady na str 646*.

Příklad 1

```
SiConnect AnyDevice;
! Receive data cyclic with 64 ms rate
SiGetCyclic AnyDevice, DataIn, 64;
```

Příklad ukazuje, jak založit spojení k externímu zařízení a nastavit cyklický přenos od zařízení `AnyDevice`.

Argumenty

```
SiGetCyclic Sensor Data Rate
```

`Sensor`

Datový typ: `sensor`

Popisovač pro externí zařízení k přijetí cyklických dat od něj. Argumentem je perzistentní proměnná a její jméno musí být stejné jako jméno určené jako klient v souboru pro nastavení *Settings.xml*.

`Data`

Datový typ: `anytype`

Reference k perzistentu obsahujícímu data pro přijetí od klienta určeného v argumentu `Sensor`. Proměnná musí být definována jako *Readable* v souboru *Configuration.xml*.

`Rate`

Datový typ: `num`

Přenosová rychlost v milisekundách (podporovány jsou pouze násobky 4 ms).

Vykonávání programu

Instrukce `SiGetCyclic` objednává data pro cyklický přenos z externího zařízení.

U instrukcí `SiGetCyclic` a `SiSetCyclic` přenosová rychlost 0 zastavuje (ruší registraci / ruší objednání) cyklický přenos daných dat nebo datové sady.

Další příklady

Více příkladů jak používat instrukci `SiGetCyclic` je názorně uvedeno dole.

Příklad 1

```
PERS sensor AnyDevice;
PERS robdata DataOut := [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
PERS sensdata DataIn :=
     ["No",[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
VAR num SampleRate:=64;
...
! Setup Interface Procedure
PROC RRI_Open()
  SiConnect AnyDevice;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, SampleRate;
  SiSetCyclic AnyDevice, DataOut, SampleRate;
ENDPROC
```

Při volání rutiny `RRI_Open` **je nejprve otevřeno spojení k zařízení se jménem** `AnyDevice`**. Potom je spuštěn cyklický přenos rychlostí** `SampleRate`**.**

Syntaxe

```
SiGetCyclic
  [ Sensor ':=' ] < persistent (PERS) of sensor > ','
  [ Data ':=' ] < persistent (PERS) of anytype > ','
  [ Rate ':=' ] < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Založit spojení k externímu systému.	Kontrola informačních štítků
Blízké spojení k externímu systému.	Kontrola informačních štítků
Data registru pro cyklický přenos.	Kontrola informačních štítků
Popisovač k externímu zařízení.	Kontrola informačních štítků
Stav komunikace zařízení.	Kontrola informačních štítků
Robot Reference Interface	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.233 SingArea - Definuje interpolaci kolem singulárních bodů

Použití

`SingArea` se používá k definování, jak se robot bude pohybovat v blízkosti singulárních bodů.

`SingArea` se také používá k definování lineární a kruhové interpolace u robotů s méně než šesti osami, a šestiosý robot může být naprogramován pro provoz s osou 4 uzamčenou na nule nebo +- 180 stupních.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklady názorně ukazují instrukci `SingArea`:

Příklad 1

```
SingArea \Wrist;
```

Orientaci nástroje je možné mírně měnit, aby přešla singulárním bodem (osy 4 a 5 v linii).

Roboty s méně než šesti osami nemusí být schopny dosáhnout interpolované orientace nástroje. Pomocí `SingArea \Wrist` může robot dosáhnout pohybu, ale orientace nástroje bude mírně změněna.

Příklad 2

```
SingArea \LockAxis4;
```

Šestiosý robot může být naprogramován k provozu s osou 4 uzamčenou na nule nebo +- 180 stupních, aby se vyloučily problémy se singularitou, když osa 5 je blízko nuly.

Naprogramované pozice bylo dosaženo s osou 4 uzamčenou na nule nebo +- 180 stupních. Jestliže pozice nebyla naprogramována s osou 4 na nule nebo +- 180 stupních, je jí nyní dosaženo s odlišnou orientací nástroje.

Jestliže se počáteční pozice osy 4 odchyluje o více než 2 stupně od uzamčené pozice, potom se první pohyb bude chovat jako když `SingArea` bylo voláno s argumentem `\Wrist`.

Osa 4 zůstane uzamčena pro všechny následné pohyby až do vykonání nové instrukce `SingArea`.

Příklad 3

```
SingArea \Off;
```

U orientace nástroje není dovolena odchylka od naprogramované orientace. Jestliže je překročen singulární bod, potom jedna nebo více os mohou provést vychylovací pohyb, jehož výsledkem je snížení rychlosti.

Roboty s méně než šesti osami nemusejí být schopné dosáhnout naprogramované orientace nástroje. Výsledkem je zastavení robotu.

Argumenty

```
SingArea [\Wrist]|[\LockAxis4]|[\Off]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

[\Wrist]

Datový typ: switch

Je dovoleno, aby se orientace nástroje poněkud lišila kvůli zabránění singularitě zápěstí. Používá se, když osy 4 a 6 jsou souběžné (osa 5 na 0 stupních). Používá se také u lineární a kruhové interpolace robotů s méně než šesti osami, kde je dovoleno odlišení orientace nástroje.

[\LockAxis4]

Datový typ: switch

Naprogramované pozice bylo dosaženo s osou 4 uzamčenou na nule nebo +- 180 stupních. Jestliže pozice nebyla naprogramována s osou 4 na nule nebo +- 180 stupních, je jí nyní dosaženo s odlišnou orientací nástroje.

Jestliže se počáteční pozice osy 4 odchyluje o více než 2 stupně od uzamčené pozice, potom se první pohyb bude chovat jako když SingArea bylo voláno s argumentem \Wrist.

[\Off]

Datový typ: switch

Není dovoleno, aby se orientace nástroje lišila. Používá se, když nejsou přecházeny žádné singulární body nebo když není dovolena změna orientace.

Jestliže žádný z argumentů není určen, systém bude nastaven na \Off.

Vykonávání programu

Jestliže je určen argument \Wrist, potom je orientace společně interpolována, aby se předešlo singulárním bodům. Tímto způsobem TCP následuje správnou dráhu, ale orientace nástroje se poněkud odchyluje. To se také stane, když singulární bod není přejet.

Jestliže je určen argument \LockAxis4, potom osa 4 je uzamčena na 0 nebo +- 180 stupních, abychom se vyhnuli singulárním bodům. TCP následuje správnou dráhu, ale orientace nástroje se bude odchylovat, jestliže pozice nebyla naprogramována s osou 4 na nule nebo +- 180 stupních.

Určená interpolace se vztahuje na všechny následné pohyby až do vykonání nové instrukce SingArea.

Pohyb je ovlivněn pouze při vykonávání lineární nebo kruhové interpolace.

Podle výchozího nastavení používá vykonávání programu automaticky argument Off pro roboty se šesti osami. Roboty s méně než šesti osami mohou použít buď argument Off nebo argument /Wrist jako standard. To je automaticky nastaveno v událostní rutině SYS_RESET.

Výchozí hodnota se nastaví automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k main
- při posunu ukazatele programu k rutině

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.233 SingArea - Definuje interpolaci kolem singulárních bodů

RobotWare - OS

Pokračování

- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Syntaxe

```
SingArea
  [ '\' Wrist ] | ['\' LockAxis4 ] | ['\' Off ] ';'
```

Související informace

Pro informace o	Viz
Singularita	*Technická referenční příručka - Přehled RAPID*
Interpolace	*Technická referenční příručka - Přehled RAPID*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.234 SiSetCyclic - Rozhraní senzoru nastaveno cyklicky

Použití

`SiSetCyclic` registruje data pro cyklický přenos na externí zařízení.

Základní příklady

Základní příklad instrukce `SiSetCyclic` je názorně uveden dole.

Viz také *Další příklady na str 651*.

Příklad 1

```
PERS sensor AnyDevice;
PERS robdata DataOut := [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
...
  SiConnect AnyDevice;
  SiSetCyclic AnyDevice, DataOut, 40;
```

Založit spojení k zařízení nazvanému `AnyDevice`. Potom registrovat data pro cyklický přenos k externímu zařízení `AnyDevice` každých 40 ms.

Argumenty

```
SiSetCyclic Sensor Data Rate
```

`Sensor`

Datový typ: `sensor`

Popisovač pro externí zařízení, ke kterému budou odeslána data.

`Data`

Datový typ: `anytype`

Reference k perzistentu komplexního nebo podporovaného jednoduchého typu obsahujícímu data pro odeslání ke klientu určenému v argumentu `Sensor`. Proměnná musí být definována jako *Writable* v souboru *Configuration.xml*.

`Rate`

Datový typ: `num`

Přenosová rychlost v milisekundách (podporovány jsou pouze násobky 4 ms).

Vykonávání programu

Instrukce `SiSetCyclic` registruje data pro cyklický přenos k externímu zařízení.

U instrukcí `SiGetCyclic` a `SiSetCyclic` přenosová rychlost 0 zastavuje (ruší registraci / ruší objednání) cyklický přenos daných dat nebo datové sady.

Další příklady

Více příkladů jak používat instrukci `SiSetCyclic` je názorně uvedeno dole.

Příklad 1

```
PERS sensor AnyDevice;
PERS robdata DataOut := [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
PERS sensdata DataIn :=
     ["No",[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
VAR num SampleRate:=64;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.234 SiSetCyclic - Rozhraní senzoru nastaveno cyklicky

Robor Reference Interface

Pokračování

```
...
! Setup Interface Procedure
PROC RRI_Open()
  SiConnect AnyDevice;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, SampleRate;
  SiSetCyclic AnyDevice, DataOut, SampleRate;
ENDPROC
```

Při volání rutiny `RRI_Open` **je nejprve otevřeno spojení k zařízení se jménem** `AnyDevice`**. Potom je spuštěn cyklický přenos rychlostí** `SampleRate`**.**

Syntaxe

```
SiSetCyclic
  [ Sensor ':=' ] < persistent (PERS) of sensor > ','
  [ Data ':=' ] < persistent (PERS) of anytype >
  [ Rate ':=' ] < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Založit spojení k externímu systému.	Kontrola informačních štítků
Blízké spojení k externímu systému.	Kontrola informačních štítků
Objednat přenos cyklických dat.	Kontrola informačních štítků
Popisovač k externímu zařízení.	Kontrola informačních štítků
Stav komunikace zařízení.	Kontrola informačních štítků
Robot Reference Interface	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.235 SkipWarn - Přeskočit poslední varování

Použití

`SkipWarn`(*Skip Warning*) se používá k přeskočení naposledy generované varovné zprávy pro uložení v protokolu událostí během vykonávání v provozním režimu nepřetržitě nebo cyklicky (žádná varování přeskočena v kroku FWD nebo BWD).

Se `SkipWarn` je možné opakovaně provádět obnovu po chybě v RAPIDu bez plnění protokolu událostí pouze varovnými zprávami.

Základní příklady

Následující příklad názorně ukazuje instrukci `SkipWarn`:

Příklad 1

```
  %"notexistingproc"%;
  nextinstruction;
ERROR
  IF ERRNO = ERR_REFUNKPRC THEN
    SkipWarn;
    TRYNEXT;
  ENDIF
ENDPROC
```

Program vykoná `nextinstruction` a žádná varovná zpráva nebude uložena do protokolu událostí.

Syntaxe

```
SkipWarn ';'
```

Související informace

Pro informace o	Viz
Obnovení po chybě	*Technická referenční příručka - Přehled RAPID* *Technická referenční příručka - Přehled RAPID*
Číslo chyby	*errnum - Chybové číslo na str 1495*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.236 SocketAccept - Přijmout příchozí spojení

Použití

`SocketAccept` se používá k přijetí požadavků na příchozí spojení. `SocketAccept` může být použito pouze pro aplikace serveru.

Základní příklady

Následující příklad názorně ukazuje instrukci `SocketAccept`:

Viz také *Další příklady na str 655*.

Příklad 1

```
VAR socketdev server_socket;
VAR socketdev client_socket;
...
SocketCreate server_socket;
SocketBind server_socket,"192.168.0.1", 1025;
SocketListen server_socket;
SocketAccept server_socket, client_socket;
```

Serverový socket je vytvořen a omezen na port `1025` na síťové adrese řadiče `192.168.0.1`. Po vykonání `SocketListen` začíná serverový socket poslouchat příchozí spojení na tomto portu a adrese. `SocketAccept` čeká na všechna příchozí spojení, přijímá požadavek na spojení a vrací klientský socket pro navázané spojení.

Argumenty

```
SocketAccept Socket ClientSocket [\ClientAddress] [ \Time ]
```

`Socket`

Datový typ: `socketdev`

Serverový socket, který čeká na příchozí spojení. Socket už musí být vytvořen, navázán a připraven na poslech.

`ClientSocket`

Datový typ: `socketdev`

Vrácený socket nového klienta, který bude aktualizován s přijatým požadavkem na příchozí spojení.

`[\ClientAddress]`

Datový typ: `string`

Proměnná, která bude aktualizována s IP adresou přijatého požadavku na příchozí spojení.

`[\Time]`

Datový typ: `num`

Max množství času [s], kdy vykonávání programu čeká na příchozí spojení. Jestliže tento čas vyprší před jakýmkoliv příchozím spojením, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_SOCK_TIMEOUT`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže se nepoužívá parametr \Time, doba čekání je 60 sek. Chcete-li čekat stále, použijte předdefinovanou konstantu WAIT_MAX.

Vykonávání programu

Serverový socket bude čekat na jakýkoliv požadavek na příchozí spojení. Když je požadavek na příchozí spojení přijat, instrukce je připravena a vrácený klientský socket je standardně připojen a může se používat v instrukcích SocketSend a SocketReceive

Další příklady

Více příkladů instrukce SocketAccept je názorně uvedeno dole.

Příklad 1

```
VAR socketdev server_socket;
VAR socketdev client_socket;
VAR string receive_string;
VAR string client_ip;
...
SocketCreate server_socket;
SocketBind server_socket, "192.168.0.1", 1025;
SocketListen server_socket;
WHILE TRUE DO
  SocketAccept server_socket, client_socket
       \ClientAddress:=client_ip;
  SocketReceive client_socket \Str := receive_string;
  SocketSend client_socket \Str := "Hello client with ip-address
       " +client_ip;
  ! Wait for client acknowledge
  ...
  SocketClose client_socket;
ENDWHILE
ERROR
  RETRY;
UNDO
  SocketClose server_socket;
  SocketClose client_socket;
```

Serverový socket je vytvořen a navázán na port 1025 na síťové adrese řadiče 192.168.0.1. Po vykonání SocketListen začíná serverový socket poslouchat příchozí spojení na tomto portu a adrese. SocketAccept přijme příchozí spojení od některého klienta a uloží klientovu adresu do řetězce client_ip. Potom server přijme řetězcovou zprávu od klienta a uloží zprávu do receive_string. Potom server odpoví zprávou " Hello client with ip-address xxx.xxx.x.x" a uzavře spojení klienta.

Potom je server připraven na spojení od stejného nebo jiného klienta ve smyčce WHILE. Jestliže se PP posune do main v programu, potom jsou všechny otevřené sockety zavřeny (SocketClose může být vždy provedeno i když socket nebyl vytvořen).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.236 SocketAccept - Přijmout příchozí spojení

Socket Messaging

Pokračování

Řešení chyb

Následující odstranitelné chyby vznikají v chybovém handleru a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_SOCK_CLOSED	Socket je zavřen (byl zavřen nebo není vytvořen). Použijte SocketCreate pro vytvoření nového socketu.
ERR_SOCK_TIMEOUT	Spojení nebylo založeno během určeného času

Syntaxe

```
SocketAccept
  [ Socket ':=' ] < variable (VAR) of socketdev > ','
  [ ClientSocket ':=' ] < variable (VAR) of socketdev >
  [ '\' ClientAddress ':=' < variable (VAR) of string> ]
  [ '\' Time ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*, sekce *Socketové odesílání zpráv*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Připojte se ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*
Odeslat data ke vzdálenému počítači	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Přijmout data od vzdáleného počítače	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Zavřít zásuvku	*SocketClose - Zavřít socket na str 659*
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad klientské socketové aplikace	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Příklad serverové socketové aplikace	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.237 SocketBind - Navázat socket k mé IP adrese a portu

Použití

`SocketBind` se používá k navázání socketu ke konkrétní IP adrese a číslu portu. `SocketBind` se může používat pouze pro serverové aplikace.

Základní příklady

Následující příklad názorně ukazuje instrukci `SocketBind`:

Příklad 1

```
VAR socketdev server_socket;

SocketCreate server_socket;
SocketBind server_socket, "192.168.0.1", 1025;
```

Serverový socket je vytvořen a navázán na port `1025` na síťové adrese řadiče `192.168.0.1`. Serverový socket je nyní možné používat v instrukci `SocketListen` pro poslech příchozích spojení na tomto portu a adrese.

Argumenty

```
SocketBind Socket LocalAddress LocalPort
```

`Socket`

Datový typ: `socketdev`

Serverový socket k navázání. Socket musí být vytvořen, ale nikoliv již navázán.

`LocalAddress`

Datový typ: `string`

Síťová adresa serveru, ke které bude socket navázán. Pouze platné jsou jakékoliv veřejné WAN adresy nebo adresa servisního portu řadiče 192.168.125.1.

`LocalPort`

Datový typ: `num`

Číslo portu serveru, ke kterému bude navázán socket. Obecně jsou volné k použití porty 1025-4999.

Vykonávání programu

Serverový socket je navázán k určenému portu serveru a IP adrese.

Jestliže určený port se již používá, bude se generovat chyba.

Použijte instrukce `SocketBind` a `SocketListen` na začátku programu k propojení lokální adresy se socketem a potom poslouchejte příchozí spojení na určeném portu. Doporučuje se provést to pouze jednou pro každý socket a port, který se používá (TCP/IP).

Použijte instrukci `SocketBind` při přijímání dat s `SocketReceiveFrom` (UDP/IP).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.237 SocketBind - Navázat socket k mé IP adrese a portu

Socket Messaging

Pokračování

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_SOCK_CLOSED	Socket je zavřen (byl zavřen nebo není vytvořen). Použijte SocketCreate k vytvoření nového socketu.
ERR_SOCK_ADDR_INUSE	Adresa a port jsou již obsazeny a není možné je použít znovu. Použijte jiné číslo portu.

Syntaxe

```
SocketBind
  [ Socket ':=' ] < variable (VAR) of socketdev > ','
  [ LocalAddress ':=' ] < expression (IN) of string > ','
  [ LocalPort ':=' ] < expression (IN) of num > ';'
```

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Připojte se ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*
Odeslat data ke vzdálenému počítači	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Přijmout data od vzdáleného počítače	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Zavřít zásuvku	*SocketClose - Zavřít socket na str 659*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad klientské socketové aplikace	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Příklad serverové socketové aplikace	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Přijmout data od vzdáleného počítače	*SocketReceiveFrom - Přijmout data od vzdáleného počítače na str 673*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.238 SocketClose - Zavřít socket

Použití

`SocketClose` se použije, když socketové připojení se už dále nebude využívat.

Když byl socket zavřen, nemůže se už použít v žádném socketovém volání kromě `SocketCreate`.

Základní příklady

Následující příklad názorně ukazuje instrukci `SocketClose`:

Příklad 1

```
SocketClose socket1;
```

Socket byl zavřen a není už možné ho používat.

Argumenty

```
SocketClose Socket
```

`Socket`

Datový typ: `socketdev`

Socket, který bude zavřen.

Vykonávání programu

Socket bude zavřen a jeho přidělené zdroje budou uvolněny.

Každý socket může být kdykoliv zavřen. Socket není možné po zavření používat. Může být znovu použit pro nové spojení po volání na `SocketCreate`.

Omezení

Zavření socketového spojení okamžitě po odeslání dat s `SocketSend` může vést ke ztrátě odesílaných dat. Je to proto, že socket `TCP/IP` má vestavěnou funkčnost pro znovuodesílání dat v případě nějakého komunikačního problému.

Chcete-li se vyhnout takovým problémům se ztrátou dat, před `SocketClose` proveďte následující:

- vyjednání vypnutí nebo
- `WaitTime` 2

Vyhněte se rychlým smyčkám s `SocketCreate ... SocketClose`, protože socket není skutečně zavřen až do určitého času (funkčnost TCP/IP).

Syntaxe

```
SocketClose
  [ Socket ':=' ] < variable (VAR) of socketdev > ';'
```

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*, sekce *Socketové odesílání zpráv*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.238 SocketClose - Zavřít socket

Socket Messaging

Pokračování

Pro informace o	Viz
Připojit ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*
Odeslat data ke vzdálenému počítači	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Přijmout data od vzdáleného počítače	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*t
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad klientské socketové aplikace	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Odeslat data ke vzdálenému počítači	*SocketSendTo - Odeslat data ke vzdálenému počítači na str 681*
Příklad serverové socketové aplikace	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Přijmout data od vzdáleného počítače	*SocketReceiveFrom - Přijmout data od vzdáleného počítače na str 673*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.239 SocketConnect - Připojit ke vzdálenému počítači

Použití

`SocketConnect` se používá k připojení socketu ke vzdálenému počítači v klientské aplikaci.

Základní příklady

Následující příklad názorně ukazuje instrukci `SocketConnect`:

Viz také *Další příklady na str 662*.

Příklad 1

```
SocketConnect socket1, "192.168.0.1", 1025;
```

Pokus o připojení ke vzdálenému počítači na IP adrese `192.168.0.1` a portu `1025`.

Argumenty

```
SocketConnect Socket Address Port [\Time]
```

`Socket`

Datový typ: `socketdev`

Klientský socket k navázání. Socket musí být vytvořen, ale nikoliv již navázán.

`Address`

Datový typ: `string`

Adresa vzdáleného počítače. Vzdálený počítač musí být určen jako IP adresa. Není možné používat jméno vzdáleného počítače.

`Port`

Datový typ: `num`

Port na vzdáleném počítači. Obecně jsou porty 1025-4999 volné k použití. Porty pod 1025 mohou být již obsazeny.

`[ \Time ]`

Datový typ: `num`

Max množství času [s], kdy vykonávání programu čeká na přijetí nebo odmítnutí spojení. Jestliže tento čas vyprší před splněním podmínky, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_SOCK_TIMEOUT`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

Jestliže se nepoužívá parametr `\Time`, doba čekání je 60 sek. Chcete-li čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Vykonávání programu

Socket se zkouší připojit ke vzdálenému počítači na určené adrese a portu. Vykonávání programu bude čekat buď na navázání spojení, selhání nebo vypršení času.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.239 SocketConnect - Připojit ke vzdálenému počítači

Socket Messaging
Pokračování

Další příklady

Více příkladů instrukce `SocketConnect` je názorně uvedeno dole.

Příklad 1

```
VAR num retry_no := 0;
VAR socketdev my_socket;
...
SocketCreate my_socket;
SocketConnect my_socket, "192.168.0.1", 1025;
...
ERROR
  IF ERRNO = ERR_SOCK_TIMEOUT THEN
    IF retry_no < 5 THEN
      WaitTime 1;
      retry_no := retry_no + 1;
      RETRY;
    ELSE
      RAISE;
    ENDIF
  ENDIF
```

Socket je vytvořen a pokusí se připojit ke vzdálenému počítači. Jestliže spojení nebude navázáno ve výchozím stanoveném čase, tj. 60 sekund, potom se pokusí připojit chybový handler. Budou provedeny čtyři pokusy a potom bude k uživateli hlášena chyba.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_SOCK_CLOSED`	Socket je zavřen (byl zavřen nebo není vytvořen). Použijte `SocketCreate` k vytvoření nového socketu.
`ERR_SOCK_TIMEOUT`	Spojení nebylo navázáno během určeného času

Syntaxe

```
SocketConnect
  [ Socket ':=' ] < variable (VAR) of socketdev > ','
  [ Address ':=' ] < expression (IN) of string > ','
  [ Port ':=' ] < expression (IN) of num >
  [ '\' Time ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Popsáno v:
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Odeslat data ke vzdálenému počítači	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Přijmout data od vzdáleného počítače	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Popsáno v:
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad klientské socketové aplikace	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Příklad serverové socketové aplikace	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.240 SocketCreate - Vytvořit nový socket

Použití

`SocketCreate` se používá pro vytvoření nového socketu pro komunikaci na základě spojení nebo bez spojení.

Je podporováno odesílání zpráv pomocí socketu streamového typu protokolu TCP/IP se zárukou doručení a datagramový protokol UDP/IP. Vytvořena může být serverová i klientská aplikace. U datagramového protokolu UDP/IP je podporováno vysílání.

Základní příklady

Následující příklad názorně ukazuje instrukci `SocketCreate`:

Příklad 1

```
VAR socketdev socket1;
...
SocketCreate socket1;
```

Nové socketové zařízení využívající protokol TCP/IP streamového typu bylo vytvořeno a přiděleno do proměnné `socket1`.

Příklad 2

```
VAR socketdev udp_sock1;
...
SocketCreate udp_sock1 \UDP;
```

Nové socketové zařízení využívající datagramový protokol TCP/IP bylo vytvořeno a přiděleno do proměnné `udp_sock1`.

Argumenty

```
SocketCreate Socket [\UDP]
```

`Socket`

Datový typ: `socketdev`

Proměnná pro uložení dat interního socketu systému.

`[ \UDP ]`

Datový typ: `switch`

Určuje, že socket by měl být typu datagramového protokolu UDP/IP.

Vykonávání programu

Instrukce vytváří nové socketové zařízení.

Socket ještě nesmí být používán. Socket je používán mezi `SocketCreate` a `SocketClose`.

Omezení

Jakýkoliv počet socketů může být deklarován, ale je možné používat současně jen 32 socketů.

Vyhněte se rychlým smyčkám s `SocketCreate ... SocketClose`, protože socket není skutečně zavřen až do určitého času (při využívání funkčnosti TCP/IP).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
SocketCreate
  [ Socket ':=' ] < variable (VAR) of socketdev >
  [ '\' UDP ] ';'
```

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*, sekce *Socketové odesílání zpráv*
Připojte se ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*
Odeslat data ke vzdálenému počítači	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Přijmout data od vzdáleného počítače	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Zavřít zásuvku	*SocketClose - Zavřít socket na str 659*
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad klientské socketové aplikace	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Odeslat data ke vzdálenému počítači	*SocketSendTo - Odeslat data ke vzdálenému počítači na str 681*
Příklad serverové socketové aplikace	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Přijmout data od vzdáleného počítače	*SocketReceiveFrom - Přijmout data od vzdáleného počítače na str 673*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.241 SocketListen - Poslouchat příchozí spojení

Použití

`SocketListen` se používá pro zahájení poslechu příchozích spojení, tj. začátek působení jako server. `SocketListen` se může používat pouze pro serverové aplikace.

Základní příklady

Následující příklad názorně ukazuje instrukci `SocketListen`:

Příklad 1

```
VAR socketdev server_socket;
VAR socketdev client_socket;
...
SocketCreate server_socket;
SocketBind server_socket, "192.168.0.1", 1025;
SocketListen server_socket;
WHILE listening DO;
  ! Waiting for a connection request
  SocketAccept server_socket, client_socket;
```

Serverový socket je vytvořen a navázán na port `1025` na síťové adrese řadiče `192.168.0.1`. Po vykonání `SocketListen` začne serverový socket poslouchat příchozí spojení na tomto portu a adrese.

Argumenty

```
SocketListen Socket
```

`Socket`

Datový typ: `socketdev`

Serverový socket, který měl začít poslouchat příchozí spojení. Socket už musí být vytvořen a navázán.

Vykonávání programu

Serverový socket začíná poslouchat příchozí spojení. Když je instrukce připravena, socket je připraven přijímat příchozí spojení.

Použijte instrukce `SocketBind` a `SocketListen` na začátku programu k propojení lokální adresy se socketem a potom poslouchejte příchozí spojení na určeném portu. Doporučuje se provést to pouze jednou pro každý socket a port, který se používá.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_SOCK_CLOSED`	Socket je zavřen (byl zavřen nebo není vytvořen). Použijte `SocketCreate` k vytvoření nového socketu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
SocketListen
  [ Socket ':=' ] < variable (VAR) of socketdev > ';'
```

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Připojte se ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*
Odeslat data ke vzdálenému počítači	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Přijmout data od vzdáleného počítače	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Zavřít zásuvku	*SocketClose - Zavřít socket na str 659*
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad klientské socketové aplikace	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Příklad serverové socketové aplikace	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.242 SocketReceive - Přijmout data od vzdáleného počítače

Použití

`SocketReceive` se používá pro příjem dat ze vzdáleného počítače.
`SocketReceive` se může použít jak pro klientské, tak i pro serverové aplikace.

Základní příklady

Následující příklad názorně ukazuje instrukci `SocketReceive`:

Viz také *Další příklady na str 670*.

Příklad 1

```
VAR string str_data;
...
SocketReceive socket1 \Str := str_data;
```

Přijmout data ze vzdáleného počítače a uložit je do řetězcové (string) proměnné `str_data`.

Argumenty

```
SocketReceive Socket [ \Str ] | [ \RawData ] | [ \Data ]
     [\ReadNoOfBytes] [\NoRecBytes] [\Time]
```

`Socket`

Datový typ: `socketdev`

V klientské aplikaci, kde socket přijímá data, musí být socket již vytvořen a připojen.

V serverové aplikaci, kde socket přijímá data, musí být socket již přijat.

`[ \Str ]`

Datový typ: `string`

Proměnná, ve které by měla být přijatá `string` data uložena. Může být zpracováno max 80 znaků.

`[ \RawData ]`

Datový typ: `rawbytes`

Proměnná, ve které by měla být přijatá rawbytes data uložena. Může být zpracováno max 1024 `rawbytes`.

`[ \Data ]`

Datový typ: `array of byte`

Proměnná, ve které by měla být přijatá bajtová data uložena. Může být zpracováno max 1024 `byte`.

Pouze jeden z volitelných parametrů `\Str`, `\RawData`, a `\Data` se může použít ve stejný okamžik.

`[ \ReadNoOfBytes ]`

Read number of Bytes

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Počet bajtů k přečtení. Min hodnota bajtů ke čtení je 1 a max množství je hodnota velikosti použitého datového typu, tj. 80 bajtů, jestliže se používá proměnná datového typu `string`.

Při komunikaci s klientem, který vždy odesílá pevný počet bajtů, může být použit tento volitelný parametr, aby bylo určeno, že stejné množství bajtů by mělo být přečteno u každé instrukce `SocketReceive`.

Jestliže odesílatel posílá RawData, příjemce potřebuje určit, že 4 bajty by měly být přijaty za každý odeslaný `rawbytes`.

`[ \NoRecBytes ]`

Number Received Bytes

Datový typ: `num`

Proměnná pro uložení počtu bajtů potřebných od určeného `socketdev`.

Stejného výsledku může být dosaženo také s

- funkcí `StrLen` na proměnné v argumentu `\Str`
- funkcí `RawBytesLen` na proměnné v argumentu `\RawData`

`[ \Time ]`

Datový typ: `num`

Max množství času [s], kdy vykonávání programu čeká na přijetí dat. Jestliže tento čas vyprší před přenosem dat, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_SOCK_TIMEOUT`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

Jestliže se nepoužívá parametr `\Time`, doba čekání je 60 sek. Chcete-li čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Vykonávání programu

Vykonání `SocketReceive` bude čekat, dokud data nebudou k dispozici nebo nedojde k selhání s chybou vypršení času.

Množství přečtených bajtů je určeno datovým typem použitým v instrukci. Při použití datového typu `string` pro příjem dat je přijato 80 bajtů, jestliže může být přečteno 80 bajtů. Při použití volitelného argumentu `ReadNoOfBytes` může uživatel určit, kolik bajtů by mělo být přijato pro každý `SocketReceive`.

Data přenášená kabelem jsou vždy bajty, max 1024 bajtů v jedné zprávě. Standardní podoba zprávy je bez hlavičky. Použití hlavičky je rezervováno pro aktuální aplikaci.

Parametr	Vstupní data	Kabelová data	Výstupní data
`\Str`	1 znak	1 bajt (8 bitů)	1 znak
`\RawData`	1 rawbytes	1 bajt (8 bitů)	1 rawbytes
`\Data`	1 bajt	1 bajt (8 bitů)	1 bajt

Je možné směšovat použitý datový typ (`string`, `rawbytes`, nebo `array of byte`) mezi `SocketSend` a `SocketReceive`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.242 SocketReceive - Přijmout data od vzdáleného počítače
Socket Messaging
Pokračování

Další příklady

Více příkladů instrukce `SocketReceive` je názorně uvedeno dole.

Příklad 1

```
VAR socketdev server_socket;
VAR socketdev client_socket;
VAR string client_ip;

PROC server_messaging()
  VAR string receive_string;
  ...
  ! Create, bind, listen and accept of sockets in error handlers
  SocketReceive client_socket \Str := receive_string;
  SocketSend client_socket \Str := "Hello client with
       ip-address"+client_ip;
  ! Wait for acknowlegde from client
  ...
  SocketClose server_socket;
  SocketClose client_socket;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=SOCK_CLOSED THEN
    server_recover;
    RETRY;
  ELSE
  ! No error recovery handling
  ENDIF
ENDPROC

PROC server_recover()
  SocketClose server_socket;
  SocketClose client_socket;
  SocketCreate server_socket;
  SocketBind server_socket, "192.168.0.1", 1025;
  SocketListen server_socket;
  SocketAccept server_socket,
       client_socket\ClientAddress:=client_ip;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    RETURN;
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC
```

Toto je příklad serverového programu s vytvořením, navázáním, poslechem a přijetím socketů v chybových handlerech. Touto cestou může program ošetřit restart po výpadku napájení.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

V proceduře `server_recover` je vytvořen serverový socket a navázán na port `1025` na síťové adrese řadiče `192.168.0.1`. Po vykonání `SocketListen` začíná serverový socket poslouchat příchozí spojení na tomto portu a adrese. `SocketAccept` přijme příchozí spojení od některého klienta a uloží adresu klienta do řetězce `client_ip`.

V komunikační proceduře `server_messaging` server přijímá řetězcovou zprávu od klienta a ukládá zprávu do `receive_string`. Potom server odpovídá se zprávou `"Hello client with ip-address xxx.xxx.x.x"`.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_SOCK_CLOSED`	Socket je zavřen. Přerušené spojení.
`ERR_SOCK_TIMEOUT`	Ve stanoveném čase nebyla přijata žádná data..

Omezení

V socketovém desílání zpráv není vestavěn žádný synchronizační mechanismus, který by vyloučil přijaté zprávy, které jsou složené z několika odeslaných zpráv. Je na programátorovi, aby vyřešil synchronizaci s „Ack“ zprávami (jedna sekvence `SocketSend - SocketReceive` v klientském nebo serverovém programu musí být dokončena před další sekvencí `SocketSend - SocketReceive`).

Všechny sockety jsou zavřeny po restartu po výpadku napájení. Tento problém může být vyřešen obnovou po chybě. Viz příklad nahoře.

Vyhněte se rychlým smyčkám s `SocketCreate ... SocketClose`, protože socket není skutečně zavřen až do určitého času (funkčnost TCP/IP).

Maximální velikost dat, která lze přijmout v jednom volání, je omezena na 1024 bajtů.

Syntaxe

```
SocketReceive
  [ Socket ':=' ] < variable (VAR) of socketdev >
  [ '\' Str ':=' < variable (VAR) of string > ]
  | [ '\' RawData ':=' < variable (VAR) of rawbytes > ]
  | [ '\' Data ':=' < array {*} (VAR) of byte > ]
  [ '\' ReadNoOfBytes ':=' < expression (IN) of num > ]
  [ '\' NoRecBytes ':=' < variable (VAR) of num > ]
  [ '\' Time ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Připojte se ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.242 SocketReceive - Přijmout data od vzdáleného počítače

Socket Messaging

Pokračování

Pro informace o	Viz
Odeslat data ke vzdálenému počítači	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Zavřít zásuvku	*SocketClose - Zavřít socket na str 659*
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad klientské socketové aplikace	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Test přítomnosti dat na socketu.	*SocketPeek - Test přítomnosti dat na socketu na str 1320*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.243 SocketReceiveFrom - Přijmout data od vzdáleného počítače

Použití

`SocketReceiveFrom` se používá pro příjem dat ze vzdáleného počítače.
`SocketReceiveFrom` se může použít jak pro klientské, tak i pro serverové aplikace.
`SocketReceiveFrom` se používá pro komunikaci bez připojení s datagramovým protokolem UDP/IP.

Základní příklady

Následující příklad názorně ukazuje instrukci `SocketReceiveFrom`:

Viz také *Další příklady na str 670*.

Příklad 1

```
VAR string str_data;
VAR string RemoteAddress;
VAR num RemotePort;
...
SocketCreate \UDP;
SocketBind myUDPsock, "192.168.9.100", 4044;
SocketReceiveFrom socket1 \Str := str_data, RemoteAddress,
     RemotePort;
```

Přijmout data od vzdáleného počítače a uložit je do řetězcové proměnné `str_data`. Adresa vzdáleného počítače se uloží do řetězcové proměnné `RemoteAddress` a číslo portu je uloženo do num proměnné `RemotePort`.

Argumenty

```
SocketReceiveFrom Socket [ \Str ] | [ \RawData ] | [ \Data ]
     [\NoRecBytes] RemoteAddress RemotePort [\Time]
```

`Socket`

Datový typ: `socketdev`

Socketové zařízení identifikující navázaný socket.

`[ \Str ]`

Datový typ: `string`

Proměnná, ve které by měla být přijatá `string` data uložena. Může být zpracováno max 80 znaků.

`[ \RawData ]`

Datový typ: `rawbytes`

Proměnná, ve které by měla být přijatá rawbytes data uložena. Může být zpracováno max 1024 `rawbytes`.

`[ \Data ]`

Datový typ: `array of byte`

Proměnná, ve které by měla být přijatá bajtová data uložena. Může být zpracováno max 1024 `byte`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.243 SocketReceiveFrom - Přijmout data od vzdáleného počítače
Socket Messaging
Pokračování

Pouze jeden z volitelných parametrů `\Str`, `\RawData`, a `\Data` se může použít ve stejný okamžik.

`[ \NoRecBytes ]`

Number Received Bytes

Datový typ: `num`

Proměnná pro uložení počtu bajtů potřebných od určeného `socketdev`.

Stejného výsledku může být dosaženo také s

- funkcí `StrLen` na proměnné v argumentu `\Str`
- funkcí `RawBytesLen` na proměnné v argumentu `\RawData`

`RemoteAddress`

Datový typ: `string`

Řetězcová proměnná obsahující zdrojovou adresu vzdáleného počítače.

`RemotePort`

Datový typ: `num`

Num proměnná obsahující port použitý vzdáleným počítačem při odesílání datagramového balíku.

`[ \Time ]`

Datový typ: `num`

Max množství času [s], kdy vykonávání programu čeká na přijetí dat. Jestliže tento čas vyprší před přenosem dat, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_SOCK_TIMEOUT`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

Jestliže se nepoužívá parametr `\Time`, doba čekání je 60 sek. Chcete-li čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Vykonávání programu

Vykonáním `SocketReceiveFrom` se přijme datagram a uloží se zdrojová adresa a zdrojový port. Bude čekat, dokud data nebudou k dispozici nebo nedojde k selhání s chybou vypršení času.

Množství přečtených bajtů je určeno datovým typem použitým v instrukci. Při použití datového typu `string` pro příjem dat je přijato 80 bajtů, jestliže může být přečteno 80 bajtů.

Data přenášená kabelem jsou vždy bajty, max 1024 bajtů v jedné zprávě. Standardní podoba zprávy je bez hlavičky. Použití hlavičky je rezervováno pro aktuální aplikaci.

Parametr	Vstupní data	Kabelová data	Výstupní data
`\Str`	1 znak	1 bajt (8 bitů)	1 znak
`\RawData`	1 rawbytes	1 bajt (8 bitů)	1 rawbytes
`\Data`	1 bajt	1 bajt (8 bitů)	1 bajt

Je možné směšovat použitý datový typ (`string`, `rawbytes`, nebo `array of byte`) mezi `SocketSendTo` a `SocketReceiveFrom`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Více příkladů instrukce `SocketReceiveFrom` je názorně uvedeno dole.

Příklad 1

```
VAR socketdev udp_socket;
VAR string client_ip;
VAR num client_port;

PROC server_messaging()
  VAR string receive_string;
  ...
  ! Create and bind of sockets in error handlers
  SocketReceiveFrom udp_socket \Str := receive_string, client_ip,
       client_port;
  SocketSendTo udp_socket, client_ip, client_port \Str := "Hello
       client with ip-address"+client_ip;
  ...
  SocketClose udp_socket;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=SOCK_CLOSED THEN
    messaging_recover;
    RETRY;
  ELSE
  ! No error recovery handling
  ENDIF
ENDPROC

PROC messaging_recover()
  SocketClose udp_socket;
  SocketCreate udp_socket \UDP;
  SocketBind udp_socket, "192.168.0.1", 1025;
ERROR
  IF ERRNO=ERR_SOCK_CLOSED THEN
    RETURN;
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC
```

Toto je příklad serverového programu s vytvořením a navázáním socketů v chybových handlerech. Touto cestou může program ošetřit restart po výpadku napájení.

V komunikační proceduře `server_messaging` server přijímá řetězcovou zprávu od klienta a ukládá zprávu do `receive_string`. Potom server odpovídá se zprávou `"Hello client with ip-address xxx.xxx.x.x"`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.243 SocketReceiveFrom - Přijmout data od vzdáleného počítače

Socket Messaging
Pokračování

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_SOCK_CLOSED	Socket je zavřen.
ERR_SOCK_TIMEOUT	Ve stanoveném čase nebyla přijata žádná data..

Omezení

Všechny sockety jsou zavřeny po restartu po výpadku napájení. Tento problém může být vyřešen obnovou po chybě. Viz příklad nahoře.

Maximální velikost dat, která lze přijmout v jednom volání, je omezena na 1024 bajtů.

Syntaxe

```
SocketReceiveFrom
  [ Socket ':=' ] < variable (VAR) of socketdev >
  [ '\' Str ':=' < variable (VAR) of string > ]
  | [ '\' RawData ':=' < variable (VAR) of rawbytes > ]
  | [ '\' Data ':=' < array {*} (VAR) of byte > ]
  [ '\' NoRecBytes ':=' < variable (VAR) of num > ]
  [ RemoteAddress ':=' ] < variable (VAR) of string >
  [ RemotePort ':=' ] < variable (VAR) of num >
  [ '\' Time ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Připojte se ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*
Odeslat data ke vzdálenému počítači	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Zavřít zásuvku	*SocketClose - Zavřít socket na str 659*
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad klientské socketové aplikace	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Odeslat data ke vzdálenému počítači	*SocketSendTo - Odeslat data ke vzdálenému počítači na str 681*
Test přítomnosti dat na socketu.	*SocketPeek - Test přítomnosti dat na socketu na str 1320*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.244 SocketSend - Odeslat data ke vzdálenému počítači

Použití

`SocketSend` se používá pro odeslání dat ke vzdálenému počítače. `SocketSend` se může použít jak pro klientské, tak i pro serverové aplikace.

Základní příklady

Následující příklad názorně ukazuje instrukci `SocketSend`:

Viz také *Další příklady na str 678*.

Příklad 1

```
SocketSend socket1 \Str := "Hello world";
```

Odesílá zprávu `"Hello world"` ke vzdálenému počítači.

Argumenty

```
SocketSend Socket [ \Str ] | [ \RawData ] | [ \Data] [ \NoOfBytes
    ]
```

`Socket`

Datový typ: `socketdev`

V klientské aplikaci, odkud socket odesílá, musí být socket již vytvořen a připojen.

V serverové aplikaci, do které socket odesílá, musí být již přijat.

`[ \Str ]`

Datový typ: `string`

`string` k odeslání ke vzdálenému počítači.

`[ \RawData ]`

Datový typ: `rawbytes`

Data `rawbytes` k odeslání ke vzdálenému počítači.

`[ \Data ]`

Datový typ: `array of byte`

Data v poli `byte` k odeslání ke vzdálenému počítači.

Pouze jeden z volitelných parametrů `\Str`,`\RawData`, nebo `\Data` se může použít ve stejný okamžik.

`[ \NoOfBytes ]`

Datový typ: `num`

Jestliže je určen tento argument, pouze tento počet bajtů bude odeslán ke vzdálenému počítači. Volání k `SocketSend` selže, jestliže `\NoOfBytes` je větší než aktuální počet bajtů v datové struktuře pro odeslání.

Jestliže tento argument není určen, potom celá datová struktura (platná část `rawbytes`) bude odeslána ke vzdálenému počítači.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.244 SocketSend - Odeslat data ke vzdálenému počítači

Socket Messaging
Pokračování

Vykonávání programu

Určená data jsou odeslána ke vzdálenému počítači. Jestliže se spojení přeruší, bude generována chyba.

Data přenášená kabelem jsou vždy bajty, max 1024 bajtů v jedné zprávě. Standardní podoba zprávy je bez hlavičky. Použití hlavičky je rezervováno pro aktuální aplikaci.

Parametr	Vstupní data	Kabelová data	Výstupní data
\Str	1 znak	1 bajt (8 bitů)	1 znak
\RawData	1 rawbytes	1 bajt (8 bitů)	1 rawbytes
\Data	1 bajt	1 bajt (8 bitů)	1 bajt

Je možné směšovat použitý datový typ (`string`, `rawbytes`, nebo `array of byte`) mezi `SocketSend` a `SocketReceive`.

Další příklady

Více příkladů instrukce `SocketSend` je názorně uvedeno dole.

Příklad 1

```
VAR socketdev client_socket;
VAR string receive_string;

PROC client_messaging()
  ...
  ! Create and connect the socket in error handlers
  SocketSend client_socket \Str := "Hello server";
  SocketReceive client_socket \Str := receive_string;
  ...
  SocketClose client_socket;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    client_recover;
    RETRY;
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC

PROC client_recover()
  SocketClose client_socket;
  SocketCreate client_socket;
  SocketConnect client_socket, "192.168.0.2", 1025;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    RETURN;
  ELSE
    ! No error recovery handling
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  ENDIF
ENDPROC
```

Toto je příklad klientského programu s vytvořením a připojením socketu v chybových handlerech. Touto cestou může program ošetřit restart po výpadku napájení.

V proceduře `client_recover` je vytvořen klientský socket a připojen k serveru vzdáleného počítače s IP adresou `192.168.0.2` na portu `1025`.

V komunikační proceduře `client_messaging` klient odesílá `"Hello server"` k serveru a server odpovídá s `"Hello client"` ke klientovi, který je uložen do proměnné `receive_string`.

Příklad 2

```
VAR socketdev client_socket;
VAR string receive_string;

PROC client_messaging()
  ...
  ! Send cr and lf to the server
  SocketSend client_socket \Str := "\0D\0A";
  ...
ENDPROC
```

Toto je příklad klientského programu, který odesílá netisknutelné znaky (binární data) v řetězci. Může to být užitečné při komunikaci se senzory nebo jinými klienty, kteří vyžadují takové znaky.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_SOCK_CLOSED`	`The socket is closed. Broken connection.`

Omezení

V socketovém odesílání zpráv není vestavěn žádný synchronizační mechanismus, který by vyloučil přijaté zprávy, které jsou složené z několika odeslaných zpráv. Je na programátorovi, aby vyřešil synchronizaci s „Ack“ zprávami (jedna sekvence `SocketSend - SocketReceive` v klientském nebo serverovém programu musí být dokončena před další sekvencí `SocketSend - SocketReceive`).

Všechny sockety jsou zavřeny po restartu po výpadku napájení. Tento problém může být vyřešen obnovou po chybě. Viz příklad nahoře.

Vyhněte se rychlým smyčkám s `SocketCreate ... SocketClose`, protože socket není skutečně zavřen až do určitého času (funkčnost TCP/IP).

Velikost dat pro odesílání je omezena na 1024 bajtů.

Syntaxe

```
SocketSend
  [ Socket ´:=´ ] < variable (VAR) of socketdev >
  [ \Str ´:=´ < expression (IN) of string > ]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.244 SocketSend - Odeslat data ke vzdálenému počítači

Socket Messaging

Pokračování

```
| [ \RawData ´:=´ < variable (VAR) of rawdata > ]
| [ \Data ´:=´ < array {*} (IN) of byte > ]
[ '\' NoOfBytes ´:=´ < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Připojte se ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*
Přijmout data od vzdáleného počítače	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Zavřít zásuvku	*SocketClose - Zavřít socket na str 659*
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad serverové socketové aplikace	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Použití netisknutelných znaků (binární data) v řetězcových literálech.	*Technická referenční příručka - RAPID kernel*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.245 SocketSendTo - Odeslat data ke vzdálenému počítači

Použití

`SocketSendTo` se používá pro odeslání dat ke vzdálenému počítače.
`SocketSendTo` se může použít jak pro klientské, tak i pro serverové aplikace.
`SocketSendTo` se používá pro komunikaci bez připojení s datagramovým protokolem UDP/IP.

Základní příklady

Následující příklad názorně ukazuje instrukci `SocketSendTo`:

Viz také *Další příklady na str 678*.

Příklad 1

```
VAR socketdev udp_socket;

SocketCreate udp_socket \UDP;
SocketSendTo udp_socket, Address, Port \Str := "Hello world";
```

Odesílá zprávu `"Hello world"` ke vzdálenému počítači s IP adresou `Address` a portem `Port`.

Argumenty

```
SocketSendTo Socket RemoteAddress RemotePort [ \Str ] | [ \RawData
     ] | [ \Data] [ \NoOfBytes ]
```

`Socket`

Datový typ: `socketdev`

Socket již musí být vytvořen.

`RemoteAddress`

Datový typ: `string`

Adresa vzdáleného počítače. Vzdálený počítač musí být určen jako IP adresa. Není možné používat jméno vzdáleného počítače.

`RemotePort`

Datový typ: `num`

Port na vzdáleném počítači. Obecně jsou porty 1025-4999 volné k použití. Porty pod 1025 mohou být již obsazeny.

`[ \Str ]`

Datový typ: `string`

`string` k odeslání ke vzdálenému počítači.

`[ \RawData ]`

Datový typ: `rawbytes`

Data `rawbytes` k odeslání ke vzdálenému počítači.

`[ \Data ]`

Datový typ: `array of byte`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.245 SocketSendTo - Odeslat data ke vzdálenému počítači

Socket Messaging
Pokračování

Data v poli `byte` k odeslání ke vzdálenému počítači.

Pouze jeden z volitelných parametrů `\Str`,`\RawData`, nebo `\Data` se může použít ve stejný okamžik.

`[ \NoOfBytes ]`

Datový typ: `num`

Jestliže je určen tento argument, pouze tento počet bajtů bude odeslán ke vzdálenému počítači. Volání k `SocketSendTo` selže, jestliže `\NoOfBytes` je větší než aktuální počet bajtů v datové struktuře pro odeslání.

Jestliže tento argument není určen, potom celá datová struktura (platná část `rawbytes`) bude odeslána ke vzdálenému počítači.

Vykonávání programu

Určená data se odesílají ke vzdálenému počítači.

Data přenášená kabelem jsou vždy bajty, max 1024 bajtů v jedné zprávě. Standardní podoba zprávy je bez hlavičky. Použití hlavičky je rezervováno pro aktuální aplikaci.

Parametr	Vstupní data	Kabelová data	Výstupní data
`\Str`	1 znak	1 bajt (8 bitů)	1 znak
`\RawData`	1 rawbytes	1 bajt (8 bitů)	1 rawbytes
`\Data`	1 bajt	1 bajt (8 bitů)	1 bajt

Je možné směšovat použitý datový typ (`string`, `rawbytes`, nebo `array of byte`) mezi `SocketSendTo` a `SocketReceiveFrom`.

Další příklady

Více příkladů instrukce `SocketSendTo` je názorně uvedeno dole.

Příklad 1

```
VAR socketdev client_socket;
VAR string receive_string;
VAR string RemoteAddress;
VAR num RemotePort;

PROC client_messaging()
  ...
  ! Create and bind the socket in error handlers
  SocketSendTo client_socket, "192.168.0.2", 1025 \Str := "Hello
       server";
  SocketReceiveFrom client_socket \Str := receive_string,
       RemoteAddress, RemotePort;
  ...
  SocketClose client_socket;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    client_recover;
    RETRY;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC

PROC client_recover()
  SocketClose client_socket;
  SocketCreate client_socket \UDP;
  SocketBind client_socket, "192.168.0.2", 1025;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    RETURN;
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC
```

Toto je příklad klientského programu s vytvořením a navázáním socketu v chybových handlerech. Touto cestou může program ošetřit restart po výpadku napájení.

V proceduře `client_recover` je vytvořen klientský socket a navázán k serveru vzdáleného počítače s IP adresou `192.168.0.2` na portu `1025`.

V komunikační proceduře `client_messaging` klient odesílá `"Hello server"` k serveru a server odpovídá s `"Hello client"` ke klientovi, který je uložen do proměnné `receive_string`.

Příklad 2

```
VAR socketdev udp_socket;

PROC message_send()
  ...
  ! Send cr and lf to the server
  SocketSendTo udp_socket, "192.168.0.2", 1025 \Str := "\0D\0A";
  ...
ENDPROC
```

Toto je příklad programu, který odesílá netisknutelné znaky (binární data) v řetězci. Může to být užitečné při komunikaci se senzory nebo jinými klienty, kteří vyžadují takové znaky.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_SOCK_CLOSED`	`The socket is closed.`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.245 SocketSendTo - Odeslat data ke vzdálenému počítači

Socket Messaging

Pokračování

Omezení

Všechny sockety jsou zavřeny po restartu po výpadku napájení. Tento problém může být vyřešen obnovou po chybě. Viz příklad nahoře.

Velikost dat pro odesílání je omezena na 1024 bajtů.

Syntaxe

```
SocketSendTo
  [ Socket ':=' ] < variable (VAR) of socketdev >
  [ RemoteAddress ':=' ] < expression (IN) of string >
  [ RemotePort ':=' ] < expression (IN) of num >
  [ \Str ':=' < expression (IN) of string > ]
  | [ \RawData ':=' < variable (VAR) of rawdata > ]
  | [ \Data ':=' < array {*} (IN) of byte > ]
  [ '\' NoOfBytes ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Připojte se ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*
Přijmout data od vzdáleného počítače	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Zavřít zásuvku	*SocketClose - Zavřít socket na str 659*
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad serverové socketové aplikace	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Přijmout data od vzdáleného počítače	*SocketReceiveFrom - Přijmout data od vzdáleného počítače na str 673*
Použití netisknutelných znaků (binární data) v řetězcových literálech.	*Technická referenční příručka - RAPID kernel*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.246 SoftAct - Aktivace měkkého serva

Použití

`SoftAct` (*Soft Servo Activate*) se používá pro aktivaci tzv. „měkkého“ serva na kterékoliv ose robotu nebo externí mechanické jednotky.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* ve všech pohybových úlohách.

Základní příklady

Následující příklad názorně ukazuje instrukci `SoftAct`:

Příklad 1

```
SoftAct 3, 20;
```

Aktivace měkkého serva na ose robotu `3` s hodnotou měkkosti `20%`.

Příklad 2

```
SoftAct 1, 90 \Ramp:=150;
```

Aktivace měkkého serva na ose robotu `1` s hodnotou měkkosti `90%` a faktorem rampy `150%`.

Příklad 3

```
SoftAct \MechUnit:=orbit1, 1, 40 \Ramp:=120;
```

Aktivace měkkého serva na ose `1` pro mechanickou jednotku `orbit1` s hodnotou měkkosti `40%` a faktorem rampy `120%`.

Argumenty

```
SoftAct [\MechUnit] Axis Softness [\Ramp]
```

`[ \MechUnit ]`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky. Jestliže je tento argument vynechán, znamená to aktivaci soft serva pro určenou osu robotu a aktuální programovou úlohu.

`Axis`

Datový typ: `num`

Číslo robotu nebo externí osy pro práci s měkkým servem.

`Softness`

Datový typ: `num`

Hodnota měkkosti v procentech (0 - 100 %). 0 % označuje min. měkkost (max. tuhost) a 100 % označuje max. měkkost.

`[ \Ramp ]`

Datový typ: `num`

Faktor rampy v procentech (> = 100 %). Faktor rampy se používá pro kontrolu zapojení měkkého serva. Faktor 100 % označuje normální hodnotu; s většími

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.246 SoftAct - Aktivace měkkého serva
RobotWare - OS
Pokračování

hodnotami se měkké servo zapojuje pomaleji (delší rampa). Výchozí hodnota pro faktor rampy je 100 %.

Vykonávání programu

Měkkost se aktivuje na hodnotě určené pro aktuální osu. Hodnota měkkosti je platná pro celý pohyb až do naprogramování nové hodnoty měkkosti pro aktuální osu nebo do deaktivace měkkého serva pomocí instrukce `SoftDeact`.

Omezení

Měkké servo u kteréhokoliv robotu nebo externí osy se vždy deaktivuje při výpadku napájení. Toto omezení je možné ošetřit v uživatelském programu při restartu po výpadku napájení.

Stejná osa nesmí být aktivována dvakrát bez vložené pohybové instrukce. Měla by tedy být vyloučena následující programová sekvence. Jinak dojde k trhnutí v pohybu robotu:

```
SoftAct n , x ;
SoftAct n , y ;
```

(n = hodnoty měkkosti os robotu n, x, a y)

VAROVÁNÍ

Brzdná vzdálenost u zastavení kategorie 1 bude delší, když měkké servo je aktivní.

Syntaxe

```
SoftAct
  ['\'MechUnit ':=' < variable (VAR) of mecunit>´,´]
  [Axis ':=' ] < expression (IN) of num> ','
  [Softness':=' ] < expression (IN) of num> ´,´
  [ '\'Ramp':=' < expression (IN) of num> ]';'
```

Související informace

Pro informace o	Viz
Deaktivovat měkké servo	*SoftDeact - Deaktivace měkkého serva na str 687*
Chování se zařazeným soft servem	*Technická referenční příručka - Přehled RAPID*
Konfigurace externích os	*Application manual - Additional axes and stand alone controller*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.247 SoftDeact - Deaktivace měkkého serva

Použití

`SoftDeact` (*Soft Servo Deactivate*) se používá pro deaktivaci takzvaného „měkkého“ serva.

Základní příklady

Následující příklady názorně ukazují instrukci `SoftDeact`:

Příklad 1

```
SoftDeact;
```

Deaktivace měkkého serva na všech osách.

Příklad 2

```
SoftDeact \Ramp:=150;
```

Deaktivace měkkého serva na všech osách s faktorem rampy 150 %.

Argumenty

```
SoftDeact [\Ramp]
```

`[ \Ramp ]`

Datový typ: `num`

Faktor rampy v procentech (> = 100 %). Faktor rampy se používá pro kontrolu deaktivace měkkého serva. Faktor 100 % označuje normální hodnotu; s většími hodnotami se měkké servo deaktivuje pomaleji (delší rampa). Výchozí hodnota pro faktor rampy je 100 %.

Vykonávání programu

Měkké servo se deaktivuje u mechanických jednotek, které jsou řízeny aktuální programovou úlohou. Jestliže `SoftDeact` je provedeno od nepohybové úlohy, měkké servo je deaktivováno pro mechanickou jednotku řízenou připojenou pohybovou úlohou. Při vykonání `SoftDeact` v režimu synchronizovaného pohybu bude měkké servo deaktivováno pro všechny mechanické jednotky, které jsou synchronizovány.

Při deaktivaci měkkého serva s `SoftDeact` se robot bude pohybovat k naprogramované pozici, i když se robot pohyboval ven z pozice během aktivace měkkého serva.

Syntaxe

```
SoftDeact
  [ '\'Ramp ':=' < expression (IN) of num> ]';'
```

Související informace

Pro informace o	Viz
Aktivace měkkého serva	*SoftAct - Aktivace měkkého serva na str 685*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.248 SpeedLimAxis - Nastavit omezení rychlosti pro osu

Použití

`SpeedLimAxis` se používá pro nastavení hodnoty rychlostního limitu pro osu. Snížení rychlosti se provádí, když vstupní signál systému `LimitSpeed` je nastaven na 1. S touto instrukcí je možné nastavovat omezení rychlosti, které by mělo být později aplikováno.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* ve všech pohybových úlohách.

Základní příklady

Následující příklady názorně ukazují instrukci `SpeedLimAxis`:

Příklad 1

```
SpeedLimAxis STN_1, 1, 20;
```

Tímto se omezí rychlost na 20 stupňů/sekundu na ose 1 u mechanické jednotky `STN_1`, když je systémový vstup `LimitSpeed` nastaven na 1.

Příklad 2

```
SpeedLimAxis ROB_1, 1, 10;
SpeedLimAxis ROB_1, 2, 30;
SpeedLimAxis ROB_1, 3, 30;
SpeedLimAxis ROB_1, 4, 30;
SpeedLimAxis ROB_1, 5, 30;
SpeedLimAxis ROB_1, 6, 30;
```

Tímto se omezí rychlost na 30 stupňů/sekundu na osách 2 až 6 u mechanické jednotky `ROB_1`, když je systémový vstup `LimitSpeed` nastaven na 1.

Argumenty

```
SpeedLimAxis MechUnit AxisNo AxisSpeed
```

`MechUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`AxisNo`

Datový typ: `num`

Číslo aktuální osy pro mechanickou jednotku.

`AxisSpeed`

Datový typ: `num`

Rychlost, která by měla být aplikována. U rotační osy by rychlost měla být ve stupních/sekundu a u lineárních os by měla být v mm/s.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

`SpeedLimAxis` se používá k nastavení hodnoty rychlostního limitu pro osu u konkrétní mechanické jednotky. Snížení rychlosti není provedeno okamžitě. Hodnoty se ukládají a jsou aplikovány, když je systémový vstupní signál `LimitSpeed` nastaven na 1.

Jestliže se nepoužívá `SpeedLimAxis` pro nastavování omezení pro osu, potom bude místo toho použito rychlostní omezení pro ruční režim. Jestliže u konkrétní osy není žádoucí žádné omezení, měla by být zadána vysoká hodnota. Dále, jestliže není nastaveno žádné omezení rychlosti kontrolního bodu pomocí instrukce `SpeedLimCheckPoint`, potom bude pro omezení rychlosti kontrolního bodu použito rychlostní omezení pro ruční režim.

Jestliže systémový vstupní signál `LimitSpeed` je nastaven na 1, rychlost je snížena (rampa) na sníženou rychlost.

Jestliže systémový vstupní signál `LimitSpeed` je nastaven na 0, rychlost je snížena (rampa) na naprogramovanou rychlost použitou v aktuální pohybové instrukci.

Systémový výstupní signál `LimitSpeed` je nastaven na 1, když je dosaženo snížené rychlosti. Systémový výstupní signál `LimitSpeed` je nastaven na 0, když se rychlost začíná zvedat (rampa).

Výchozí hodnoty pro rychlostní omezení jsou nastavovány automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Další příklady

Více příkladů instrukce `SpeedLimAxis` je názorně uvedeno dole.

Příklad 1

```
..
VAR intnum sigint1;
VAR intnum sigint2;
..
PROC main()
  ! Setup interrupts reacting on a signal input
  IDelete sigint1;
  CONNECT sigint1 WITH setlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 1, sigint1;
  IDelete sigint2;
  CONNECT sigint2 WITH resetlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 0, sigint2;
  ..
  MoveL p1, z50, fine, tool2;
  MoveL p2, z50, fine, tool2;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.248 SpeedLimAxis - Nastavit omezení rychlosti pro osu
Pokračování

```
  ..
  MoveL p10, v100, fine, tool2;
  ! Set limitations for checkpoints and axes
  SpeedLimCheckPoint 200;
  SpeedLimAxis ROB_1, 1, 10;
  SpeedLimAxis ROB_1, 2, 10;
  SpeedLimAxis ROB_1, 3, 10;
  SpeedLimAxis ROB_1, 4, 20;
  SpeedLimAxis ROB_1, 5, 20;
  SpeedLimAxis ROB_1, 6, 20;
  WHILE run_loop = TRUE DO
    MoveL p1, vmax, z50, tool2;
    ..
    MoveL p99, vmax, fine, tool2;
  ENDWHILE
  ! Set the default manual mode max speed
  SpeedLimCheckPoint 0;
  SpeedLimAxis ROB_1, 1, 0;
  SpeedLimAxis ROB_1, 2, 0;
  SpeedLimAxis ROB_1, 3, 0;
  SpeedLimAxis ROB_1, 4, 0;
  SpeedLimAxis ROB_1, 5, 0;
  SpeedLimAxis ROB_1, 6, 0;
  ..
TRAP setlimitspeed
  IDelete sigint1;
  CONNECT sigint1 WITH setlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 1, sig1int1;
  ! Set out signal that is cross connected to system input
       LimitSpeed
  SetDO do1LimitSpeed, 1;
ENDTRAP
TRAP resetlimitspeed
  IDelete sigint2;
  CONNECT sigint2 WITH resetlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 0, sig1int2;
  ! Reset out signal that is cross connected to system input
       LimitSpeed
  SetDO do1LimitSpeed, 0;
ENDTRAP
```

Během pohybu robotu od pozice `p1` **k** `p10` **se použije výchozí rychlostní omezení (rychlost ručního režimu). Bude přidáno nové rychlostní omezení pro kontrolní body u TCP robotu a os.** `TRAP setlimitspeed` **bude aplikovat rychlostní omezení, jestliže signál** `mysensorsignal` **změní hodnotu na 1.**

`TRAP resetlimitspeed` **odstraní rychlostní omezení, když signál** `mysensorsignal` **změní hodnotu na 0.**

Nová nastavení pro rychlostní omezení budou použita, jakmile proměnná `run_loop` **je** `TRUE` **a systémový vstupní signál** `LimitSpeed` **je nastaven na 1. Když je**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`run_loop` nastaven na `FALSE`, nastaví se výchozí rychlostní omezení (rychlost ručního režimu).

POZNÁMKA

Rutina `TRAP` v příkladu je použita pouze k vizualizaci funkčnosti. Signál použitý k omezení rychlosti by mohl být také připojen buď přímo k systémovému vstupnímu signálu `LimitSpeed` nebo přes bezpečnostní PLC.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_AXIS_PAR`	Parametr osa v instrukci je nesprávný
`ERR_SPEEDLIM_VALUE`	Rychlost použitá v argumentu `AxisSpeed` je příliš nízká.

Omezení

`SpeedLimAxis` se nemůže používat v událostní rutině POWER ON.

Při snižování rychlosti na jedné ose nebo kontrolním bodu budou také ostatní osy omezeny o stejné procento, aby byly schopné běžet podél naprogramované dráhy. Procesní rychlost podél naprogramované dráhy se bude měnit.

Při použití SafeMove společně s omezením rychlosti musí být nastaven SafeMove s hranicí, jelikož výpočty SafeMove a pohybu jsou mírně odlišné.

Syntaxe

```
SpeedLimAxis
    [ MechUnit':=' ] < variable (VAR) of mecunit> ','
    [ AxisNo':=' ] < expression (IN) of num> ','
    [ AxisSpeed':=' ] < expression (IN) of num> ';'
```

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Nastavit rychlostní omezení pro kontrolní body	*SpeedLimCheckPoint - Nastavit rychlostní omezení pro kontrolní body na str 692*
Systémové vstupní a výstupní signály	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.249 SpeedLimCheckPoint - Nastavit rychlostní omezení pro kontrolní body

Použití

`SpeedLimCheckPoint` se používá pro nastavení hodnoty rychlostního limitu pro TCP robot. Snížení rychlosti se provádí, když vstupní signál systému `LimitSpeed` je nastaven na 1. S touto instrukcí je možné nastavovat omezení rychlosti, které by mělo být později aplikováno.

Snížení rychlosti se provádí, jestliže kterýkoliv z kontrolních bodů běží rychleji než je limit nastavený v `SpeedLimCheckPoint`. (Více informací o kontrolních bodech najdete v *Další příklady na str 694*.)

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* ve všech pohybových úlohách.

Základní příklady

Následující příklad názorně ukazuje instrukci `SpeedLimCheckPoint`:

Příklad 1

```
VAR num limit_speed:=200;
SpeedLimCheckPoint limit_speed;
```

Tímto se omezí rychlost na 200 mm/s u TCP robotu, když je systémový vstup `LimitSpeed` nastaven na 1.

Argumenty

```
SpeedLimCheckPoint RobSpeed
```

`RobSpeed`

Datový typ: `num`

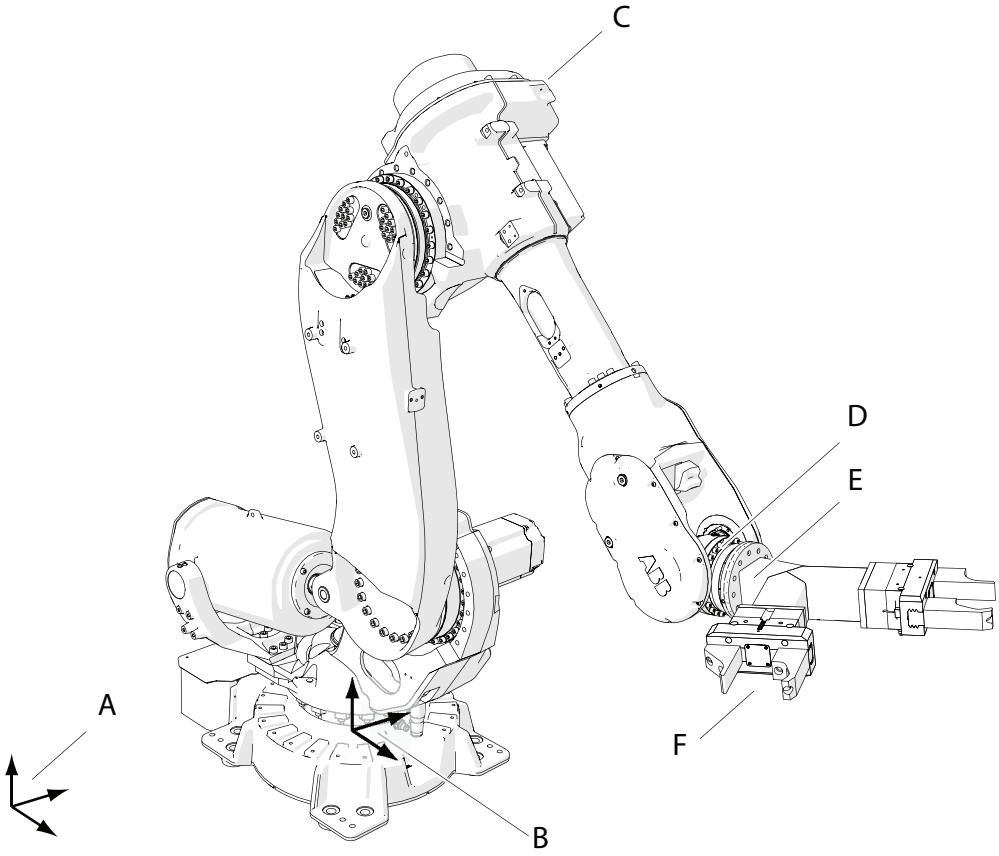
Rychlostní omezení v mm/s, které by mělo být aplikováno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Definice kontrolních bodů - viz obrázek dole.



xx1200000521

A	Světový souřadnicový systém
B	Souřadnicový systém základny
C	Kontrolní bod ramena
D	Střední bod zápěstí (WCP)
E	tool0
F	Střední bod nástroje (TCP)

`SpeedLimCheckPoint` se používá pro nastavení hodnoty rychlostního limitu pro 4 kontrolní body u TCP robotu. Kontrolní body, které budou omezeny, jsou rameno, zápěstí, tool0 a aktivní TCP, jak je vidět na obrázku nahoře. Snížení rychlosti nenastává okamžitě. Hodnoty jsou uloženy a jsou použity, když systémový vstupní signál `LimitSpeed` je nastaven na 1. Rychlost kontrolních bodů je omezena ve vztahu k souřadnému systému základny.

Jestliže se nepoužívá `SpeedLimCheckPoint` pro nastavování omezení, potom bude místo toho použito rychlostní omezení pro ruční režim. Jestliže u kontrolních bodů není žádoucí žádné omezení, měla by být zadána vysoká hodnota. Dále, jestliže není nastaveno žádné omezení rychlosti osy pomocí instrukce `SpeedLimAxis`, potom bude pro omezení rychlosti osy použito rychlostní omezení pro ruční režim.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.249 SpeedLimCheckPoint - Nastavit rychlostní omezení pro kontrolní body
Pokračování

Jestliže systémový vstupní signál `LimitSpeed` je nastaven na 1, rychlost je snížena (rampa) na sníženou rychlost.

Jestliže systémový vstupní signál `LimitSpeed` je nastaven na 0, rychlost je snížena (rampa) na naprogramovanou rychlost použitou v aktuální pohybové instrukci.

Systémový výstupní signál `LimitSpeed` je nastaven na 1, když je dosaženo snížené rychlosti. Systémový výstupní signál `LimitSpeed` je nastaven na 0, když se rychlost začíná zvedat (rampa).

Výchozí hodnoty pro rychlostní omezení jsou nastavovány automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Další příklady

Více příkladů instrukce `SpeedLimCheckPoint` je názorně uvedeno dole.

Příklad 1

```
..
VAR intnum sigint1;
VAR intnum sigint2;
..
PROC main()
  ! Setup interrupts reacting on a signal input
  IDelete sigint1;
  CONNECT sigint1 WITH setlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 1, sigint1;
  IDelete sigint2;
  CONNECT sigint2 WITH resetlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 0, sigint2;
  ..
  MoveL p1, z50, fine, tool2;
  MoveL p2, z50, fine, tool2;
  ..
  MoveL p10, v100, fine, tool2;
  ! Set limitations for checkpoints and axes
  SpeedLimCheckPoint 200;
  SpeedLimAxis ROB_1, 1, 10;
  SpeedLimAxis ROB_1, 2, 10;
  SpeedLimAxis ROB_1, 3, 10;
  SpeedLimAxis ROB_1, 4, 20;
  SpeedLimAxis ROB_1, 5, 20;
  SpeedLimAxis ROB_1, 6, 20;
  WHILE run_loop = TRUE DO
    MoveL p1, vmax, z50, tool2;
    ..
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
    MoveL p99, vmax, fine, tool2;
  ENDWHILE
  ! Set the default manual mode max speed
  SpeedLimCheckPoint 0;
  SpeedLimAxis ROB_1, 1, 0;
  SpeedLimAxis ROB_1, 2, 0;
  SpeedLimAxis ROB_1, 3, 0;
  SpeedLimAxis ROB_1, 4, 0;
  SpeedLimAxis ROB_1, 5, 0;
  SpeedLimAxis ROB_1, 6, 0;
  ..
TRAP setlimitspeed
  IDelete sigint1;
  CONNECT sigint1 WITH setlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 1, sig1int1;
  ! Set out signal that is cross connected to system input
       LimitSpeed
  SetDO do1LimitSpeed, 1;
ENDTRAP
TRAP resetlimitspeed
  IDelete sigint2;
  CONNECT sigint2 WITH resetlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 0, sig1int2;
  ! Reset out signal that is cross connected to system input
       LimitSpeed
  SetDO do1LimitSpeed, 0;
ENDTRAP
```

Během pohybu robotu od pozice `p1` k `p10` se použije výchozí rychlostní omezení (rychlost ručního režimu). Bude přidáno nové rychlostní omezení pro kontrolní body u TCP robotu a os. `TRAP setlimitspeed` bude aplikovat rychlostní omezení, jestliže signál `mysensorsignal` změní hodnotu na 1.

`TRAP resetlimitspeed` odstraní rychlostní omezení, když signál `mysensorsignal` změní hodnotu na 0.

Nová nastavení pro rychlostní omezení budou použita, jakmile proměnná `run_loop` je `TRUE` a systémový vstupní signál `LimitSpeed` je nastaven na 1. Když je `run_loop` nastaven na `FALSE`, nastaví se výchozí rychlostní omezení (rychlost ručního režimu).

POZNÁMKA

Rutina `TRAP` v příkladu je použita pouze k vizualizaci funkčnosti. Signál použitý k omezení rychlosti by mohl být také připojen buď přímo k systémovému vstupnímu signálu `LimitSpeed` nebo přes bezpečnostní PLC.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_SPEEDLIM_VALUE`	Rychlost použitá v argumentu `RobSpeed` je příliš nízká.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.249 SpeedLimCheckPoint - Nastavit rychlostní omezení pro kontrolní body

Pokračování

Omezení

`SpeedLimCheckPoint` se nemůže používat v událostní rutině POWER ON.

Jestliže robot stojí na pohyblivé trati, potom rychlost kontrolního bodu ve světovém rámci může být vyšší než určený limit rychlosti kontrolního bodu v základnovém rámci. Rychlost kontrolního bodu ve světovém rámci může být součtem traťové rychlosti a rychlosti kontrolního bodu v základnovém rámci. Také u omezení rychlosti kontrolního bodu ve světovém rámci se ujistěte, že součet těchto dvou nepřekračuje limit.

Při snižování rychlosti na jedné ose nebo kontrolním bodu budou také ostatní osy omezeny o stejné procento, aby byly schopné běžet podél naprogramované dráhy. Procesní rychlost podél naprogramované dráhy se bude měnit.

Při používání SafeMove společně s omezením rychlosti musí být SafeMove nastaven s tolerancí a otestován, jelikož SafeMove a výpočty pohybu jsou mírně odlišné. Změna TCP nástroje za chodu není synchronizována s SafeMove. Takže TCP nástroje v SafeMove musí být buď kratší než nástroje používané robotem, nebo max rychlost kontrolního bodu pro SafeMove musí být nastavena se zvláštní tolerancí a otestována.

Syntaxe

```
SpeedLimCheckPoint
  [ RobSpeed ':=' ] < expression (IN) of num > ';'
```

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Nastavit omezení rychlosti pro osu	*SpeedLimAxis - Nastavit omezení rychlosti pro osu na str 688*
Definování zátěží ramene	*Technická referenční příručka - Systémové parametry*
Systémové vstupní a výstupní signály	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.250 SpeedRefresh - Aktualizovat potlačení rychlosti pro probíhající pohyb

Použití

`SpeedRefresh` se používá pro změnu rychlosti pohybu pro probíhající pohyb robotu v aktuální programové úloze. S touto instrukcí je možné vytvořit jistý druh hrubého přizpůsobení rychlosti od vstupu některého senzoru.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* ve všech pohybových úlohách.

Základní příklady

Následující příklad názorně ukazuje instrukci `SpeedRefresh`:

Příklad 1

```
VAR num change_speed:=70;
SpeedRefresh change_speed;
```

Tímto se změní aktuální potlačení rychlosti na 70 %.

Argumenty

```
SpeedRefresh Override
```

`Override`

Datový typ: `num`

Hodnota potlačení rychlosti v rozsahu 0 ... 100 %.

Vykonávání programu

Hodnota potlačení aktuální rychlosti pro probíhající pohyby robotu a externích os v aktuální pohybové programové úloze bude aktualizována.

Všechny komponenty rychlostních dat u všech mechanických jednotek v aktuální pohybové úloze budou ovlivněny.

POZNÁMKA

Potlačení rychlosti nastavené od `SpeedRefresh` není rovné nastavení od FlexPendantu. To jsou dvě odlišné hodnoty. Produktem těchto dvou hodnot a naprogramované rychlosti bude rychlost, která je použita v pohybu.

Jestliže je provedeno PP k main nebo jestliže je načten nový program, rychlost, která byla nastavena s `SpeedRefresh`, bude resetována, a rychlost nastavená od FlexPendantu bude použita.

Další příklady

Více příkladů instrukce `SpeedRefresh` je názorně uvedeno dole.

Příklad 1

```
VAR intnum time_int;
VAR num override;
...
PROC main()
  CONNECT time_int WITH speed_refresh;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.250 SpeedRefresh - Aktualizovat potlačení rychlosti pro probíhající pohyb
RobotWare - OS
Pokračování

```
  ITimer 0.1, time_int;
  ISleep time_int;
  ...
  MoveL p1, v100, fine, tool2;
  ! Read current speed override set from FlexPendant
  override := CSpeedOverride (\CTask);
  IWatch time_int;
  MoveL p2, v100, fine, tool2;
  IDelete time_int;
  ! Reset to FlexPendant old speed override
  WaitTime 0.5;
  SpeedRefresh override;
  ...
TRAP speed_refresh
  VAR speed_corr;
  ! Analog input signal value from sensor, value 0 ... 10
  speed_corr := (ai_sensor * 10);
  SpeedRefresh speed_corr;
  ERROR
    IF ERRNO = ERR_SPEED_REFRESH_LIM THEN
      IF speed_corr > 100 speed_corr := 100;
      IF speed_corr < 0 speed_corr := 0;
      RETRY;
    ENDIF
ENDTRAP
```

Během pohybu robotu od pozice `p1` k `p2` je hodnota potlačení rychlosti aktualizována každou `0.1` sek. v `TRAP speed_refresh`. Analogový vstupní signál `ai_sensor` se používá pro výpočet hodnoty `Overide` pro instrukci `SpeedRefresh`. Nedojde k vykonání `TRAP` před a po pohybu robotu mezi `p1` a `p2`. Ruční potlačení rychlosti od FlexPendantu je obnoveno. Potom by měl robot dosáhnout `p2`.

Řešení chyb

Jestliže `Override` má hodnotu mimo rozsah 0 až 100 %, potom bude proměnná `ERRNO` nastavena na `ERR_SPEED_REFRESH_LIM`. Tato chyba je odstranitelná a může být ošetřena v chybovém handleru `ERROR`

Omezení

Všimněte si, že s `SpeedRefresh` nebude potlačení rychlosti provedeno okamžitě. Místo toho dojde ke zpoždění 0,3 - 0,5 sekund mezi příkazem a vlivem na fyzický robot.

Uživatel je odpovědný za reset hodnoty potlačení rychlosti z programu RAPID po sekvenci `SpeedRefresh`.

Jestliže se použije `SpeedRefresh` v `START` nebo v událostní rutině `RESET`, rychlost, která je nastavena, je vždy aktuální potlačení rychlosti FlexPendantu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.250 SpeedRefresh - Aktualizovat potlačení rychlosti pro probíhající pohyb

RobotWare - OS

Pokračování

Syntaxe

```
SpeedRefresh
  [ Override ':=' ] < expression (IN) of num > ';'
```

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Přečíst aktuální potlačení rychlosti	*CSpeedOverride - Čte aktuální rychlost potlačení na str 1115*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.251 SpyStart - Spustit záznam dat času vykonávání

Použití

`SpyStart` se používá pro spuštění záznamu instrukčních a časových dat během vykonávání.

Data vykonávání budou uložena do souboru pro pozdější analýzu.

Uložená data jsou určena pro odlaďování programů RAPID, zvláště pro multitaskingové systémy (pouze je nutné mít `SpyStart - SpyStop` v jedné programové úloze).

Základní příklady

Následující příklad názorně ukazuje instrukci `SpyStart`:

Příklad 1

```
SpyStart "HOME:/spy.log";
```

Začíná zaznamenávat data doby vykonávání do souboru `spy.log` na disku `HOME:`.

Argumenty

```
SpyStart File
```

Soubor

Datový typ: `string`

Cesta souboru a jméno souboru do souboru, který bude obsahovat data vykonávání.

Vykonávání programu

Určený soubor je otevřen pro zápis, a data doby vykonávání se začínají zaznamenávat do souboru.

Záznam dat doby vykonávání je aktivní až do:

- vykonávání instrukce `SpyStop`
- spuštění vykonávání programu od začátku
- načtení nového programu
- dalšího **Restartu**

`SpyStart` je ovlivněn automatickým resetem podmínky

Omezení

Vyhněte se používání diskety (doplněk) pro záznam, protože zápis na disketu je časově velmi náročný.

Nikdy nepoužívejte spy funkci ve výrobních programech, protože funkce prodlužuje čas cyklu a spotřebovává paměť na používaném paměťovém médiu.

Řešení chyb

Jestliže soubor v instrukci `SpyStart` není možné otevřít, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEOPEN` (viz „Datové typy - `errnum`"). Tato chyba může být ošetřena v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Formát souboru

ÚLOHA	INSTR	V	KÓD	OUT
MAIN	FOR i FROM 1 TO 3 DO	0	PŘIPRAVENO	0
MAIN	mynum:=mynum+i;	1	PŘIPRAVENO	1
MAIN	ENDFOR	2	PŘIPRAVENO	2
MAIN	mynum:=mynum+i;	2	PŘIPRAVENO	2
MAIN	ENDFOR	2	PŘIPRAVENO	2
MAIN	mynum:=mynum+i;	2	PŘIPRAVENO	2
MAIN	ENDFOR	2	PŘIPRAVENO	3
MAIN	SetDo1,1;	3	PŘIPRAVENO	3
MAIN	IF di1=0 THEN	3	PŘIPRAVENO	4
MAIN	MoveL p1, v1000, fine, tool0;	4	WAIT	14
MAIN	MoveL p1, v1000, fine, tool0;	111	PŘIPRAVENO	111
MAIN	ENDIF	108	PŘIPRAVENO	108
MAIN	MoveL p2, v1000, fine, tool0;	111	WAIT	118
MAIN	MoveL p2, v1000, fine, tool0;	326	PŘIPRAVENO	326
MAIN	SpyStop;	326	PŘIPRAVENO	

Sloupec ÚLOHA (TASK) ukazuje vykonávanou programovou úlohu.

Sloupec INSTR ukazuje vykonávanou instrukci v určené programové úloze.

Sloupec IN (DOVNITŘ) ukazuje čas v ms při vstupu do vykonávané instrukce.

Sloupec KÓD (CODE) ukazuje, jestli instrukce je PŘIPRAVENA (READY) nebo instrukci WAIT (ČEKAT) pro dokončení při vypršení času (OUT).

Sloupec OUT (VEN) ukazuje čas v ms při opuštění vykonávané instrukce.

Všechny časy jsou udávány v ms (relativní hodnoty).

SYSTEM TRAP znamená, že systém provádí něco jiného než vykonávání instrukcí RAPID.

Jestliže procedura volá nějakou NOSTEPIN proceduru (modul), potom výstupní seznam ukazuje pouze jméno volané procedury. To je opakováno u každé vykonávané instrukce v rutině NOSTEPIN.

Syntaxe

```
SpyStart
  [File':=']<expression (IN) of string>';'
```

Související informace

Pro informace o	Viz
Zastavit záznam dat vykonávání	*SpyStop - Zastavit záznam dat doby vykonávání na str 702*
Auto podmínka	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.252 SpyStop - Zastavit záznam dat doby vykonávání

Použití

`SpyStop` se používá pro zastavení záznamu časových dat během vykonávání.

Data, která mohou být užitečná pro optimalizaci doby vykonávání cyklu, jsou uložena do souboru pro pozdější analýzu.

Základní příklady

Následující příklad názorně ukazuje instrukci `SpyStop` :

Viz také *Další příklady na str 702*.

Příklad 1

```
SpyStop;
```

Zastavuje záznam dat doby vykonávání do souboru určeného předchozí instrukcí `SpyStart`.

Vykonávání programu

Záznam dat doby vykonávání je zastaven a soubor určený předchozí instrukcí `SpyStart` je zavřen. Jestliže žádná instrukce `SpyStart` nebyla předtím vykonávána, potom je instrukce `SpyStop` ignorována.

Další příklady

Více příkladů instrukce `SpyStop` je názorně uvedeno dole.

Příklad 1

```
IF debug = TRUE SpyStart "HOME:/spy.log";
produce_sheets;
IF debug = TRUE SpyStop;
```

Jestliže odlaďovací příznak je true, potom spusťte záznam dat vykonávání do souboru `spy.log` na disku `HOME:`. Proveďte aktuální produkci; zastavte záznam a zavřete soubor `spy.log`.

Omezení

Vyhněte se používání diskety (doplněk) pro záznam, protože zápis na disketu je časově velmi náročný.

Nikdy nepoužívejte spy funkci ve výrobních programech, protože funkce prodlužuje čas cyklu a spotřebovává paměť na používaném paměťovém médiu.

Syntaxe

```
SpyStop’;’
```

Související informace

Pro informace o	Viz
Spustit záznam dat vykonávání	*SpyStart - Spustit záznam dat času vykonávání na str 700*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.253 StartLoad - Načíst programový modul během vykonávání

Použití

`StartLoad` se používá pro zahájení načítání programového modulu do paměti programu během vykonávání.

Když probíhá načítání, je možné provádět souběžně jiné instrukce. Načtený modul musí být připojen k programové úloze s instrukcí `WaitLoad` předtím, než jakékoliv jeho symboly/rutiny mohou být použity.

Načtený programový modul bude přidán k již existujícím modulům v paměti programu.

Programový nebo systémový modul mohou být načteny ve statickém (výchozí) nebo dynamickém režimu. Podle použitého režimu některé operace stáhnou modul nebo modul vůbec neovlivní.

Statický režim

Následující tabulka popisuje, jak dvě různé operace ovlivňují staticky načtené programové nebo systémové moduly.

	Nastavit PP na main z TP	Otevřít nový program RAPID
Programový modul	Není ovlivněno	Staženo
Systémový modul	Není ovlivněno	Není ovlivněno

Dynamický režim

Následující tabulka popisuje, jak dvě různé operace ovlivňují dynamicky načtené programové nebo systémové moduly.

	Nastavit PP na main z TP	Otevřít nový program RAPID
Programový modul	Staženo	Staženo
Systémový modul	Staženo	Staženo

Staticky a dynamicky načtené moduly mohou být staženy instrukcí `UnLoad`.

Základní příklady

Následující příklad názorně ukazuje instrukci `StartLoad`:

Viz také *Další příklady na str 705*.

Příklad 1

```
VAR loadsession load1;

! Start loading of new program module PART_B containing routine
     routine_b in dynamic mode
StartLoad \Dynamic, diskhome \File:="PART_B.MOD", load1;

! Executing in parallel in old module PART_A containing routine_a
%"routine_a"%;

! Unload of old program module PART_A
UnLoad diskhome \File:="PART_A.MOD";
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.253 StartLoad - Načíst programový modul během vykonávání

RobotWare - OS

Pokračování

```
! Wait until loading and linking of new program module PART_B is
     ready
WaitLoad load1;

! Execution in new program module PART_B
%"routine_b"%;
```

Spouští načítání programového modulu `PART_B.MOD` z `diskhome` do paměti programu s instrukcí `StartLoad`. Souběžně s načítáním program vykonává `routine_a` v modulu `PART_A.MOD`. Potom instrukce `WaitLoad` čeká na dokončení načítání a připojení. Modul je načten v dynamickém režimu.

Proměnná `load1` drží identitu načítací akce aktualizované od `StartLoad` a odkazované od `WaitLoad`.

Kvůli úspoře doby spojování může být instrukce `UnLoad` a `WaitLoad` kombinována v instrukci `WaitLoad` pomocí volitelného argumentu `\UnLoadPath`.

Argumenty

```
StartLoad [\Dynamic] FilePath [\File] LoadNo
```

`[\Dynamic]`

Datový typ: `switch`

Spínač zapíná načítání programového modulu v dynamickém režimu. Jinak běží načítání ve statickém režimu.

`FilePath`

Datový typ: `string`

Cesta souboru a jméno souboru k souboru, který bude načten do paměti programu. Jméno souboru by mělo být vyřazeno, když se používá argument `\File` .

`[\File]`

Datový typ: `string`

Když je jméno souboru vyřazeno v argumentu `FilePath`, musí být definováno s tímto argumentem.

`LoadNo`

Datový typ: `loadsession`

Toto je reference k načítací akci, která by měla být použita v instrukci `WaitLoad` kvůli připojení načteného programového modulu k programové úloze.

Vykonávání programu

Vykonání `StartLoad` pouze přikáže načítání a potom pokračuje přímo s další instrukcí bez čekání na dokončení načítání.

Instrukce `WaitLoad` bude potom čekat, nejprve na dokončení načítání, pokud nebylo dosud dokončeno, a potom bude propojena a inicializována. Iniciace načteného modulu nastavuje všechny proměnné na úrovni modulu na jejich počáteční hodnoty.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Nevyřešené reference budou vždy akceptovány pro tuto operaci načítání `StartLoad - WaitLoad`, ale bude to chyba při běhu při vykonávání nevyřešené reference.

Aby bylo možné získat dobrou programovou strukturu, která je snadná na porozumění i udržování, veškeré načítání a stahování programových modulů by se mělo provádět od hlavního modulu, který je vždy přítomen v paměti programu během vykonávání.

Pro načítání programu, který obsahuje proceduru `main` k hlavnímu programu (s další procedurou `main`), viz instrukce `Load`, *Load - Načíst programový modul během provádění na str 328*.

Další příklady

Více příkladů jak používat instrukci `StartLoad` je názorně uvedeno dole.

Příklad 1

```
StartLoad \Dynamic, "HOME:/DOORDIR/DOOR1.MOD", load1;
```

Načítá programový modul `DOOR1.MOD` z `HOME:` v adresáři `DOORDIR` do paměti programu. Programový modul je načten v dynamickém režimu.

Příklad 2

```
StartLoad \Dynamic, "HOME:" \File:="/DOORDIR/DOOR1.MOD", load1;
```

Stejné jako v příkladu 1, ale s jinou syntaxí.

Příklad 3

```
StartLoad "HOME:" \File:="/DOORDIR/DOOR1.MOD", load1;
```

Stejné jako v příkladech 1 a 2 nahoře, ale modul je načten ve statickém režimu.

Příklad 4

```
StartLoad \Dynamic, "HOME:" \File:="/DOORDIR/DOOR1.MOD", load1;
WaitLoad load1;
```

je stejné jako

```
Load \Dynamic, "HOME:" \File:="/DOORDIR/DOOR1.MOD";
```

Řešení chyb

Jestliže soubor určený v instrukci již není možné najít, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILNOTFND`. Tato chyba může být potom ošetřena v chybovém handleru.

Jestliže proměnná určená v `LoadNo` se již používá, potom je systémová proměnná `ERRNO` nastavena na `ERR_LOADNO_INUSE`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
StartLoad
  [´\´Dynamic ´,´]
  [FilePath’ :=’] <expression (IN) of string>
  [’\’File ’:=’ <expression (IN) of string> ] ’,’
  [LoadNo ’:=’] <variable (VAR) of loadsession>’;’
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.253 StartLoad - Načíst programový modul během vykonávání

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Připojit načtený modul k úloze	*WaitLoad - Připojit načtený modul k úloze na str 947*
Načíst akci	*loadsession - Načtení programu na str 1530*
Načíst programový modul	*Load - Načíst programový modul během provádění na str 328*
Zrušit načtení programového modulu	*UnLoad - Stáhnout programový modul během provádění na str 905*
Zrušit načítání programového modulu	*CancelLoad - Zrušit načítání modulu na str 64*
Volání procedury s opožděnou vazbou	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.254 StartMove - Znovu spouští pohyb robotu

Použití

`StartMove` se používá pro obnovu pohybu robotu, externích os a souvisejícího procesu po zastavení pohybu

- instrukcí `StopMove`.
- po vykonání sekvence `StorePath` ... `RestoPath`.
- po asynchronně pozvednutých chybách pohybů, jako je `ERR_PATH_STOP` nebo specifická procesní chyba po zpracovávání v chybovém handleru `ERROR`.

U základnového systému je možné používat tuto instrukci v následujícím typu programových úloh:

- hlavní úloha `T_ROB1` pro restart pohybu v této úloze.
- každá další úloha pro restart pohybů v hlavní úloze.

U systému MultiMove je možné používat tuto instrukci v následujícím typu programových úloh:

- pohybová úloha, pro restart pohybu v této úloze.
- nepohybová úloha, pro restart pohybu v připojené pohybové úloze. Kromě toho, jestliže pohyb je restartován v jedné připojené pohybové úloze příslušející ke skupině koordinovaných synchronizovaných úloh, pohyb je restartován ve všech spolupracujících úlohách.

Základní příklady

Následující příklady názorně ukazují instrukci `StartMove`:

Příklad 1

```
StopMove;
WaitDI ready_input,1;
StartMove;
```

Robot se začíná znovu pohybovat, když je nastaven vstup `ready_input`.

Příklad 2

```
...
MoveL p100, v100, z10, tool1;
StorePath;
p:= CRobT(\Tool:=tool1);
! New temporary movement
MoveL p1, v100, fine, tool1;
...
MoveL p, v100, fine, tool1;
RestoPath;
StartMove;
...
```

Po pohybu zpět na pozici zastavení `p` (v tomto příkladu rovné s `p100`), se robot začíná znovu pohybovat na základní úrovni dráhy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.254 StartMove - Znovu spouští pohyb robotu
Pokračování

Argumenty

```
StartMove [\AllMotionTasks]
```

`[\AllMotionTasks]`

Datový typ: `switch`

Restartovat pohyb všech mechanických jednotek v systému. Přepínač `[\AllMotionTasks]` může být použit pouze od nepohybové programové úlohy.

Vykonávání programu

Všechny procesy spojené se zastaveným pohybem jsou restartovány ve stejném okamžiku, kdy se pohyb obnoví.

Pro restartování aplikace MultiMove v synchronizovaném koordinovaném režimu musí být vykonán `StartMove` ve všech pohybových úlohách, které jsou zapojeny do koordinace.

S přepínačem `\AllMotionTasks` (povoleno pouze z nepohybové programové úlohy) jsou restartovány pohyby všech mechanických jednotek v systému.

V základnovém systému bez přepínače `\AllMotionTasks` jsou restartovány pohyby u následujících mechanických jednotek:

- vždy mechanické jednotky v hlavní úloze, nezávisle na tom, která úloha vykonává instrukci `StartMove`.

V systému MultiMove bez přepínače `\AllMotionTasks` jsou restartovány pohyby u následujících mechanických jednotek:

- mechanické jednotky v pohybové úloze vykonávající `StartMove`.
- mechanické jednotky v pohybové úloze, které jsou připojeny k nepohybové úloze vykonávající `StartMove`. Kromě toho, jestliže mechanické jednotky jsou restartovány v jedné připojené pohybové úloze příslušející ke skupině koordinovaných synchronizovaných úloh, potom jsou mechanické jednotky restartovány ve všech spolupracujících úlohách.

Řešení chyb

Jestliže robot je příliš daleko od dráhy (více než 10 mm nebo 20 stupňů), než aby mohl provést restart přerušeného pohybu, potom je systémová proměnná `ERRNO` nastavena na `ERR_PATHDIST`.

Jestliže robot je ve stavu zastavení v době, kdy je vykonáván `StartMove`, potom je systémová proměnná `ERRNO` nastavena na `ERR_STARTMOVE`. `ERR_STARTMOVE` může být také způsoben nouzovým zastavením, jestliže je vhodné načasování.

Jestliže je vykonávání programu zastaveno několikrát během obnovování pohybu na dráze s `StartMove`, potom je systémová proměnná `ERRNO` nastavena na `ERR_PROGSTOP`.

Jestliže se robot pohybuje v době, kdy je vykonáván `StartMove`, potom je systémová proměnná `ERRNO` nastavena na `ERR_ALRDY_MOVING`.

Tyto chyby mohou být potom zpracovány v chybovém handleru:

- u `ERR_PATHDIST` posuňte robot blíž ke dráze před pokusem o `RETRY`.
- u `ERR_STARTMOVE`, `ERR_PROGSTOP` nebo `ERR_ALRDY_MOVING` počkejte nějakou dobu před pokusem o `RETRY`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Pouze jedné z několika nepohybových úloh je povoleno provést ve stejnou dobu sekvenci `StopMove` - `StartMove` proti nějaké pohybové úloze.

Není možné provádět žádnou obnovu po chybě, jestliže je vykonáván `StartMove` v jakémkoliv chybovém handleru.

Syntaxe

```
StartMove
  ['\'AllMotionTasks]';'
```

Související informace

Pro informace o	Viz
Zastavení pohybů	*StopMove - Zastavuje pohyby robotu na str 736*
Pokračování v pohybu	*StartMoveRetry - Restartuje pohyb robotu a vykonávání na str 710*
Další příklady	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742* *RestoPath - Obnovuje cestu po přerušení na str 546*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.255 StartMoveRetry - Restartuje pohyb robotu a vykonávání

Použití

`StartMoveRetry` se používá k obnovení pohybů robotu a externích os a souvisejících procesů a také k dalšímu pokusu o provedení z chybového handleru `ERROR`.

Tuto instrukci je možné používat v chybovém handleru `ERROR` v následujících typech programových úloh:

- hlavní úloha `T_ROB1` v základnovém systému
- každá pohybová úloha v systému *MultiMove*

Základní příklady

Následující příklad názorně ukazuje instrukci `StartMoveRetry`:

Příklad 1

```
VAR robtarget p_err;
...
MoveL p1\ID:=50, v1000, z30, tool1 \WObj:=stn1;
...
ERROR
  IF ERRNO = ERR_PATH_STOP THEN
    StorePath;
    p_err := CRobT(\Tool:= tool1 \WObj:=wobj0);
    ! Fix the problem
    MoveL p_err, v100, fine, tool1;
    RestoPath;
    StartMoveRetry;
  ENDIF
ENDPROC
```

Toto je příklad ze systému MultiMove s koordinovanými synchronizovanými pohyby (dva roboty pracující na jistém otáčeném pracovním objektu).

Během pohybu k pozici `p1` jiný spolupracující robot dostává jistou procesní chybu, takže se koordinované synchronizované pohyby zastaví. Tento robot potom dostane chybu `ERR_PATH_STOP` a vykonávání je přeneseno k chybovém handleru `ERROR`.

V chybovém handleru `ERROR` proveďte následující:

- `StorePath` ukládá původní dráhu, přechází na úroveň nové dráhy a nastavuje systém MultiMove do nezávislého režimu.
- Jestliže se vyskytují problémy s robotem, potom iniciujte pohyby na úrovni nové dráhy.
- Před `RestoPath` přejděte zpět na pozici chyby.
- `RestoPath` přechází zpět na úroveň původní dráhy a nastavuje systém MultiMove zpět do synchronizovaného režimu.
- `StartMoveRetry` restartuje přerušený pohyb a jakýkoliv proces. Také přenáší vykonávání zpět, aby se normální vykonávání obnovilo.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

`StartMoveRetry` provádí následující sekvenci:

- návrat na dráhu
- restartovat všechny procesy spojené se zastaveným pohybem
- restartovat přerušený pohyb
- `RETRY` vykonávání programu

`StartMoveRetry` provádí to samé jako `StartMove` a `RETRY` společně s jednou nedělitelnou operací.

Jsou restartovány pouze mechanické jednotky v programové úloze, která vykonává `StartMoveRetry`.

Omezení

Může být použit pouze v chybovém handleru `ERROR` v pohybové úloze.

V systému MultiMove musí být při vykonávání koordinovaných synchronizovaných pohybů respektována následující programovací pravidla v chybovém handleru `ERROR`.

- `StartMoveRetry` musí být použito ve všech spolupracujících programových úlohách.
- Jestliže potřebujeme pohyb v některém chybovém handleru `ERROR`, potom musíme použít instrukce `StorePath ... RestoPath` ve všech spolupracujících programových úlohách.
- Program musí posunout robot zpět k pozici chyby před vykonáním `RestoPath`, jestliže robot byl posunut na úroveň `StorePath`.

Řešení chyb

Jestliže robot je příliš daleko od dráhy (více než 10 mm nebo 20 stupňů), než aby mohl provést restart přerušeného pohybu, potom je systémová proměnná `ERRNO` nastavena na `ERR_PATHDIST`.

Jestliže robot je ve stavu podržení v době, kdy je vykonáván `StartMoveRetry`, potom je systémová proměnná `ERRNO` nastavena na `ERR_STARTMOVE`.

Jestliže je vykonávání programu zastaveno několikrát během obnovování pohybu na dráze s `StartMoveRetry`, potom je systémová proměnná `ERRNO` nastavena na `ERR_PROGSTOP`..

Jestliže se robot pohybuje v době, kdy je vykonáván `StartMoveRetry`, potom je systémová proměnná `ERRNO` nastavena na `ERR_ALRDY_MOVING`.

Není možné provádět žádnou obnovu po chybě od těchto chyb, protože `StartMoveRetry` může být vykonán pouze v některém chybovém handleru.

Syntaxe

```
StartMoveRetry ';'
```

Související informace

Pro informace o	Viz
Zastavení pohybů	*StopMove - Zastavuje pohyby robotu na str 736*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.255 StartMoveRetry - Restartuje pohyb robotu a vykonávání

RobotWare - OS

Pokračování

Pro informace o	Viz
Pokračování v pohybu	*StartMove - Znovu spouští pohyb robotu na str 707*
Obnovit vykonávání po chybě	*RETRY - Obnovit vykonávání po chybě na str 548*
Uložit/obnovit dráhu	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742* *RestoPath - Obnovuje cestu po přerušení na str 546*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.256 STCalib - Kalibrovat servo nástroj

Použití

`STCalib` se používá pro kalibraci vzdálenosti mezi hroty nástroje. To je nutné po výměně hrotu nebo výměně nástroje a doporučuje se to po provedení broušení hrotu nebo po použití nástroje jen na chvíli.

Pozor! Nástroj provádí během kalibrace dva pohyby zavřít/otevřít. První zavírací pohyb zjistí kontaktní pozici hrotu.

Základní příklady

Následující příklady názorně ukazují instrukci `STCalib`:

Příklad 1

```
VAR num curr_tip_wear;
VAR num retval;
CONST num max_adjustment := 20;

STCalib gun1 \ToolChg;
```

Kalibrovat servo pistoli po výměně nástroje. Čekejte na dokončení kalibrace pistole a potom pokračujte s další instrukcí RAPID.

Příklad 2

```
STCalib gun1 \ToolChg \Conc;
```

Kalibrovat servo pistoli po výměně nástroje. Pokračujte s další instrukcí RAPID bez čekání na dokončení kalibrace pistole.

Příklad 3

```
STCalib gun1 \TipChg;
```

Kalibrovat servo pistoli po výměně hrotu.

Příklad 4

```
STCalib gun1 \TipWear \RetTipWear := curr_tip_wear;
```

Kalibrovat servo pistoli po opotřebení hrotu. Uložte opotřebení hrotu do proměnné `curr_tip_wear`.

Příklad 5

```
STCalib gun1 \TipChg \RetPosAdj:=retval;
IF retval > max_adjustment THEN
TPWrite "The tips are lost!";
...
```

Kalibrovat servo pistoli po výměně hrotu. Zkontrolujte, jestli hroty chybějí.

Příklad 6

```
STCalib gun1 \TipChg \PrePos:=10;
```

Kalibrovat servo pistoli po výměně hrotu. Proveďte rychlý pohyb do pozice `10` mm, potom začněte hledat kontaktní pozici nižší rychlostí.

Příklad 7

Příklad neplatné kombinace:

```
STCalib gun1 \TipWear \RetTipWear := curr_tip_wear \Conc;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.256 STCalib - Kalibrovat servo nástroj

Servo Tool Control

Pokračování

Provést kalibraci opotřebení hrotu. Pokračujte s další instrukcí RAPID bez čekání na dokončení kalibrace pistole. Parametr `curr_tip_wear` v tomto případě **nebude** držet žádnou platnou hodnotu, protože se používá přepínač `\Conc` (Další instrukce RAPID se začne vykonávat před dokončením kalibračního procesu).

Argumenty

```
STCalib ToolName [\ToolChg] | [\TipChg] | [\TipWear] [\RetTipWear]
     [\RetPosAdj] [\PrePos] [\Conc]
```

`ToolName`

Datový typ: `string`

Jméno mechanické jednotky.

`[\ToolChg]`

Datový typ: `switch`

Kalibrace po výměně nástroje.

`[\TipChg]`

Datový typ: `switch`

Kalibrace po výměně hrotu.

`[\TipWear]`

Datový typ: `switch`

Kalibrace po opotřebení hrotu.

`[\RetTipWear]`

Datový typ: `num`

Dosažené opotřebení hrotu [mm].

`[\RetPosAdj]`

Datový typ: `num`

Nastavení pozice od poslední kalibrace [mm].

`[\PrePos]`

Datový typ: `num`

Pozice, kam se přesunout vysokou rychlostí před zahájením hledání kontaktní pozice nižší rychlostí [mm].

`[\Conc]`

Datový typ: `switch`

Následné instrukce se vykonávají během pohybu pistole. Argument se může použít pro zkrácení doby cyklu. To je užitečné, když jsou např. dvě pistole řízeny současně.

Vykonávání programu

Kalibrační režimy

Jestliže existuje mechanická jednotka, potom je servo nástroj přikázán ke kalibraci. Kalibrace se provede podle přepínačů, viz dole. Jestliže se použije parametr `RetTipWear`, potom je opotřebení hrotu aktualizováno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Kalibrace po výměně nástroje:

Nástroj se zavře malou rychlostí a čeká na hroty v kontaktu, rychle se otevře, rychle se zavře na malou sílu a znovu otevře v jedné sekvenci. Opotřebení hrotu zůstane nezměněno.

Kalibrace po výměně hrotu:

Nástroj se zavře malou rychlostí a čeká na hroty v kontaktu, které se otevřou rychle, rychle se otevře, rychle se zavře na malou sílu a znovu otevře v jedné sekvenci. Opotřebení hrotu bude resetováno.

Kalibrace po opotřebení hrotu:

Nástroj se zavře vysokou rychlostí do kontaktní pozice, rychle se otevře, rychle se zavře na malou sílu a znovu se otevře v jedné sekvenci. Opotřebení hrotu bude aktualizováno.

POZOR! Jestliže se používá přepínač `Conc`, potom bude instrukce považována za připravenou, jakmile se spustila a proto nebude vratná hodnota `RetTipWear`k dispozici. V tomto případě bude `RetTipWear` vrácen funkcí `STIsOpen`. Další podrobnosti najdete ve funkcích RobotWareOS - `STIsOpen`.

Nastavení pozice

Volitelný argument `RetPosAdj` se může použít pro zjištění, například, jestli jsou hroty po výměně hrotu ztraceny. Parametr bude držet hodnotu nastavení pozice od poslední kalibrace. Hodnota může být záporná nebo kladná.

Používání předpozice

Kvůli urychlení kalibrace je možné definovat předpozici. Když kalibrace začne, rameno pistole se rychle přesune k předpozici, zastaví se a potom pokračuje pomalu *) dopředu, aby zjistilo kontaktní pozici hrotu. Jestliže používáte předpozici, vybírejte ji opatrně! Je důležité, aby se hroty dostaly do kontaktu až *po* dosažení předpozice! Jinak bude přesnost kalibrace špatná a mohou se objevit chyby dohledu pohybu. Předpozice bude ignorována, jestliže je větší než aktuální pozice pistole (aby se nezpomalovala kalibrace).

*) Druhý pohyb bude také rychlý, jestliže se používá doplněk `\TipWear`

Řešení chyb

Jestliže jméno určeného servo nástroje není konfigurovaný servo nástroj, potom je systémová proměnná `ERRNO` nastavena na `ERR_NO_SGUN`.

Jestliže pistole není otevřena, když je vyvoláván `STCalib`, potom je systémová proměnná `ERRNO` nastavena na `ERR_SGUN_NOTOPEN`.

Jestliže mechanická jednotka servo nástroje není aktivována, potom je systémová proměnná `ERRNO` nastavena na `ERR_SGUN_NOTACT`. Pro aktivaci servo nástroje použijte instrukci `ActUnit`.

Jestliže pozice servo nástroje není iniciována, potom je systémová proměnná `ERRNO` nastavena na `ERR_SGUN_NOTINIT`. Pozice servo nástroje musí být iniciována poprvé, když je instalována pistole nebo po provedení jemné kalibrace. Použijte servisní rutinu `ManServiceCalib` nebo proveďte kalibraci změny hrotu. Opotřebení hrotu bude resetováno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.256 STCalib - Kalibrovat servo nástroj

Servo Tool Control

Pokračování

Jestliže hroty servo nástroje nejsou synchronizovány, potom je systémová proměnná ERRNO nastavena na ERR_SGUN_NOTSYNC. Hroty servo nástroje musí být synchronizovány, jestliže počítadlo otáček bylo ztraceno a/nebo aktualizováno. Žádná procesní data, jako opotřebení hrotu, nebudou ztracena.

Jestliže instrukce je vyvolána od úlohy v pozadí a dojde k nouzovému zastavení, instrukce bude dokončena a systémová proměnná ERRNO se nastaví na ERR_SGUN_ESTOP. Všimněte si, že po vyvolání instrukce z hlavní úlohy bude ukazatel programu zastaven na instrukci a instrukce bude restartována od začátku při restartu programu.

Jestliže je určen argument PrePos s hodnotou menší než nula, potom bude systémová proměnná ERRNO nastavena na ERR_SGUN_NEGVAL.

Jestliže instrukce je vyvolána od úlohy v pozadí a systém je ve stavu vypnutých motorů, potom bude systémová proměnná ERRNO nastavena na ERR_SGUN_MOTOFF.

Všechny shora uvedené chyby mohou být zpracovány v chybovém manipulátoru RAPID.

Syntaxe

```
STCalib
  [ 'ToolName' :=' ] < expression (IN) of string > ‘,’
  [ '\'ToolChg] | ['\'TipChg] | [ '\'TipWear]
  [' \'RetTipWear' :=' < variable or persistent(INOUT) of num >
       ]';'
  [ '\'RetPosAdj' :=' < variable or persistent(INOUT) of num > ]';'
  [ '\'PrePos' :=' < expression (IN) of num > ]'
  [ '\'Conc' ];'
```

Související informace

Pro informace o	Viz
Otevřít servo nástroj	*STOpen - Otevřít servo nástroj na str 734*
Zavřít servo nástroj	*STClose - Zavřít servo nástroj na str 717*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.257 STClose - Zavřít servo nástroj

Použití

`STClose` se používá pro zavření servo nástroje.

Základní příklady

Následující příklady názorně ukazují instrukci `STClose`:

Příklad 1

```
VAR num curr_thickness1;
VAR num curr_thickness2;

STClose gun1, 1000, 5;
```

Zavřít servo nástroj se sílou hrotu `1000` N a tloušťkou desky `5` mm. Čekejte, až se pistole zavře, potom pokračujte s další instrukcí RAPID.

Příklad 2

```
STClose gun1, 2000, 3\RetThickness:=curr_thickness;
```

Zavřít servo nástroj se sílou hrotu `2000` N a tloušťkou desky `3` mm. Získejte změřenou tloušťku v proměnné `curr_thickness`.

Příklad 3

Souběžný reřim:

```
STClose gun1, 1000, 5 \Conc;
STClose gun2, 2000, 3 \Conc;
```

Zavřít servo `gun1` se sílou hrotu `1000` N a tloušťkou desky `5` mm. Pokračujte ve vykonávání programu bez čekání na zavření `gun1`, a zavřete servo `gun2` se sílou hrotu `2000`N a tloušťkou desky `3` mm. Pokračujte ve vykonávání programu RAPID bez čekání na zavření `gun2`.

Příklad 4

```
IF STIsClosed (gun1)\RetThickness:=curr_thickness1 THEN
  IF curr_thickness1 < 0.2 Set weld_start1;
ENDIF
IF STIsClosed (gun2)\RetThickness:=curr_thickness2 THEN
  IF curr_thickness2 < 0.2 Set weld_start2;
ENDIF
```

Vezměte změřenou tloušťku ve funkci `STIsClosed` proměnná `curr_thickness1` a `curr_thickness2`.

Příklad 5

Příklad neplatné kombinace:

```
STClose gun1, 2000, 3\RetThickness:=curr_thickness \Conc;
```

Zavřete servo pistoli a pokračujte s vykonáváním programu RAPID. Parametr `curr_thickness` v tomto případě nebude držet žádnou platnou hodnotu, protože se používá přepínač `\Conc` (Další instrukce RAPID se začne vykonávat před před zavřením pistole).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.257 STClose - Zavřít servo nástroj

Servo Tool Control
Pokračování

Argumenty

```
STClose ToolName TipForce Thickness [\RetThickness][\Conc]
```

`ToolName`

Datový typ: `string`

Jméno mechanické jednotky.

`TipForce`

Datový typ: `num`

Požadovaná síla hrotu [N].

`Thickness`

Datový typ: `num`

Očekávaná kontaktní pozice pro servo nástroj [mm].

`[\RetThickness]`

Datový typ: `num`

Dosažená tloušťka [mm] získá hodnotu, pouze když se nepoužívá přepínač `\Conc`.

`[\Conc]`

Datový typ: `switch`

Následné instrukce se vykonávají během pohybu pistole. Argument se může použít pro zkrácení doby cyklu. To je užitečné, když jsou např. dvě pistole řízeny současně.

Vykonávání programu

Jestliže existuje mechanická jednotka, potom je servo nástroj přikázán zavřít na očekávanou tloušťku a sílu.

Zavírání spustí pohyb ramena nástroje k očekávané kontaktní pozici (tloušťka). Pohyb je zastaven na této pozici a je provedeno přepnutí z kontrolního režimu pozice do kontrolního režimu síly.

Rameno nástroje je posunuto max rychlostí a zrychlením, jak jsou definovány v systémových parametrech pro odpovídající externí osy. U pohybů ostatních os je rychlost snížena v ručním režimu.

Když je dosaženo požadované síly hrotu, instrukce je připravena a dosažená tloušťka je vrácena, jestliže je určen volitelný argument `RetThickness`.

POZOR! Jestliže se používá přepínač `Conc`**, potom bude instrukce považována za připravenou, jakmile se spustila a proto nebude vratná hodnota** `RetThickness` **k dispozici. V tomto případě bude** `RetThickness` **vrácen funkcí** `STIsClosed`**. Další podrobnosti najdete ve funkcích RobotWareOS -** `STIsClosed`.

Je možné zavřít nástroj během naprogramovaného pohybu robotu, jelikož pohyb robotu neobsahuje pohyb ramena nástroje.

Více podrobností viz Řízení pohybu servo nástroje.

Řešení chyb

Jestliže jméno určeného servo nástroje není konfigurovaný servo nástroj, potom je systémová proměnná `ERRNO` **nastavena na** `ERR_NO_SGUN`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže pistole není otevřena, když je vyvoláván `STClose`, potom je systémová proměnná `ERRNO` nastavena na `ERR_SGUN_NOTOPEN`.

Jestliže mechanická jednotka servo nástroje není aktivována, potom je systémová proměnná `ERRNO` nastavena na `ERR_SGUN_NOTACT`. Pro aktivaci servo nástroje použijte instrukci `ActUnit`.

Jestliže pozice servo nástroje není iniciována, potom je systémová proměnná `ERRNO` nastavena na `ERR_SGUN_NOTINIT`. Pozice servo nástroje musí být iniciována poprvé, když je instalována pistole nebo po provedení jemné kalibrace. Použijte servisní rutinu `ManServiceCalib` nebo proveďte kalibraci změny hrotu. Opotřebení hrotu bude resetováno.

Jestliže hroty servo nástroje nejsou synchronizovány, potom je systémová proměnná `ERRNO` nastavena na `ERR_SGUN_NOTSYNC`. Hroty servo nástroje musí být synchronizovány, jestliže počítadlo otáček bylo ztraceno a/nebo aktualizováno. Žádná procesní data, jako opotřebení hrotu, nebudou ztracena.

Jestliže instrukce je vyvolána od úlohy v pozadí a dojde k nouzovému zastavení, instrukce bude dokončena a systémová proměnná `ERRNO` se nastaví na `ERR_SGUN_ESTOP`. Všimněte si, že po vyvolání instrukce z hlavní úlohy bude ukazatel programu zastaven na instrukci a instrukce bude restartována od začátku při restartu programu.

Jestliže instrukce je vyvolána od úlohy v pozadí a systém je ve stavu vypnutých motorů, potom bude systémová proměnná `ERRNO` nastavena na `ERR_SGUN_MOTOFF`.

Všechny shora uvedené chyby mohou být zpracovány v chybovém manipulátoru RAPID.

Syntaxe

```
STClose
  [ 'ToolName ':=' ] < expression (IN) of string > ‘,’
  [ 'Tipforce' :=' ] < expression (IN) of num > ‘,’
  [ 'Thickness' :='] < expression (IN) of num > ]
  [‘\’ 'RetThickness' :=' < variable or persistent (INOUT) of num
       > ]
  ['\'Conc]
```

Související informace

Pro informace o	Viz
Otevřít servo nástroj	*STOpen - Otevřít servo nástroj na str 734*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.258 StepBwdPath - Posunout zpět o jeden krok na dráze

Použití

`StepBwdPath` se používá pro pohyb TCP zpět na dráze robotu z událostní rutiny `RESTART`.

Je na uživateli, aby zavedl příznak procesu restartu, takže se vykoná pouze `StepBwdPath` v událostní rutině `RESTART` při restartu procesu a nikoliv při všech restartech programu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `StepBwdPath`:

Příklad 1

```
StepBwdPath 30, 1;
StepBwdPath 30, 1;
```

První instrukce se posune zpět o 30 mm. Druhá instrukce se posune zpět o dalších 30 mm.

Argumenty

```
StepBwdPath StepLength StepTime
```

`StepLength`

Datový typ: `num`

Určuje vzdálenost v milimetrech pro pohyb zpět během tohoto kroku. Tento argument musí být kladná hodnota.

`StepTime`

Datový typ: `num`

Tento argument je překonaný. Nastavte ho na 1.

Vykonávání programu

Robot se posune zpět na své dráze o určenou vzdálenost. Dráha je přesně stejná v reverzním módu, jako byla před vzniklým zastavením. V případě rychlého zastavení nebo nouzového zastavení je volána událostní rutina `RESTART` po dokončení fáze vrácení, takže robot bude již zpět na své dráze, když je tato instrukce vykonána.

Aktuální rychlost tohoto pohybu je naprogramovaná rychlost v příkazu pohybu, ale omezená na 250 mm/s.

Následující vlastnosti jsou platné v synchronizovaných koordinovaných pohybech systému MultiMove:

- Všechny zapojené mechanické jednotky jsou posunuty zpět současně a koordinovány
- Každý vykonaný `StepBwdPath` v každé zapojené programové úloze má výsledek v podobě jednoho nového kroku zpětného pohybu (bez potřeby jakéhokoliv `StartMove`)

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

- Aby pohyby přerušeného procesu byly restartovány a pokračovaly, musí být vykonána instrukce `StartMove` ve všech zapojených programových úlohách.

Omezení

Po zastavení programu je možné krokovat zpětně na dráze s následujícími limity:

- Pohyby `StepBwdPath` jsou omezeny k poslednímu jemnému bodu historie příkazů pohybu a délce historie příkazů pohybu, kterých je normálně pět.

Jestliže je učiněn pokus o pohyb za tyto limity, potom bude volán chybový handler s `ERRNO` nastavenou na `ERR_BWDLIMIT`.

Syntaxe

```
StepBwdPath
  [ StepLength':=' ] < expression (IN) of num >','
  [ StepTime ':=' ] < expression (IN) of num >';'
```

Související informace

Pro informace o	Viz
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.259 STIndGun - Nastavuje pistoli do nezávislého režimu

Použití

`STIndGun`(*Servo Tool independent gun*) se používá pro nastavení pistole do nezávislého režimu a tedy posunutí pistole do určené nezávislé pozice. Pistole zůstane v nezávislém režimu, dokud není provedena instrukce `STIndGunReset`.

Během nezávislého režimu je ovladač pistole oddělen od robotu. Pistole může být zavřena, otevřena, kalibrována nebo posunuta do nové nezávislé pozice, ale nebude následovat koordinované pohyby robotu.

Nezávislý režim je užitečný, jestliže pistole provádí úlohu, která je nezávislá na úloze robotu, např. broušení hrotu stacionární pistole.

Základní příklady

Následující příklad názorně ukazuje instrukci `STIndGun`:

Příklad 1

Tato procedura může běžet od úlohy v pozadí, zatímco robot v hlavní úloze může pokračovat např. s pohybem instrukcí.

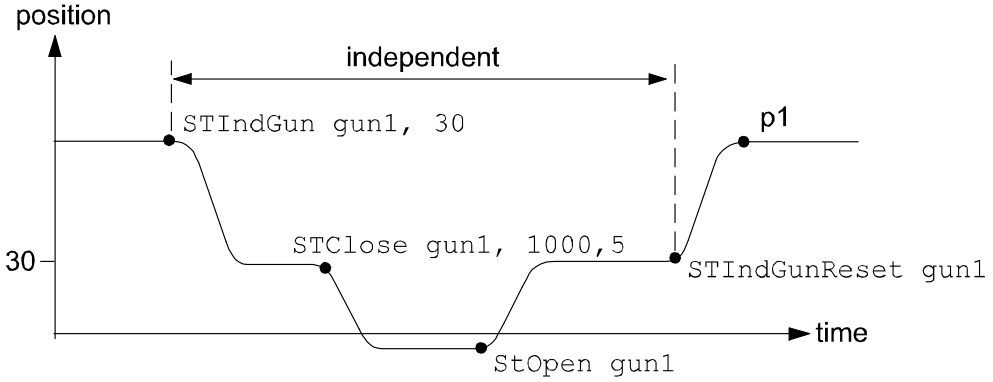
```
PROC tipdress()

  ! Note that the gun will move to current robtarget position, if
  ! already in independent mode.
  STIndGunReset gun1;
  ...
  STIndGun gun1, 30;
  StClose gun1, 1000, 5;
  WaitTime 10;
  STOpen gun1;
  ...
  STIndGunReset gun1;

ENDPROC
```

Nezávislý režim se aktivuje a pistole je posunuta do nezávislé pozice (`30` mm). Během nezávislého pohybu jsou vykonávány instrukce `StClose`, `WaitTime`, a

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

STOpen bez zasahování do pohybu robotu. Instrukce StIndGunReset vyjme pistoli z nezávislého režimu a posune pistoli do aktuální pozice robtarget.



xx0500002342

Pozice p1 závisí na pozici pistole dané v robtarget právě provedené robotem.

Argumenty

```
STIndGun ToolName GunPos
```

ToolName

Datový typ: string

Jméno mechanické jednotky.

GunPos

Datový typ: num

Pozice (zdvih) servo pistole v mm.

Syntaxe

```
STIndGun
  [ ToolName ':=' ] < expression (IN) of string > ','
  [ GunPos ':=' < expression (IN) of num > ]';'
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.260 STIndGunReset - Resetuje pistoli z nezávislého režimu

Použití

`STIndGunReset` (*Servo Tool independent gun reset*) se používá k resetu pistole z nezávislého režimu a potom posunutí pistole k aktuální pozici robtarget.

Základní příklady

Následující příklad názorně ukazuje instrukci `STIndGunReset`:

```
STIndGunReset gun1;
```

Argumenty

```
STIndGunReset ToolName
```

`ToolName`

Datový typ: `string`

Jméno mechanické jednotky.

Vykonávání programu

Instrukce bude resetovat pistoli z nezávislého režimu a posune pistoli do aktuální pozice robtarget. Během tohoto pohybu musí být koordinovaná rychlost pistole nula, jinak dojde k chybě. Koordinovaná rychlost bude nula, jestliže robot stojí v klidu nebo když aktuální pohyb robotu zahrnuje „nulový pohyb“ od pistole.

Syntaxe

```
STIndGunReset
  [ToolName ´:=´]<expression (IN) of string>´;´
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.261 SToolRotCalib - Kalibrace TCP a rotace pro stacionární nástroj

Použití

`SToolRotCalib` *(Stationary Tool Rotation Calibration)* se používá pro kalibrování TCP a otáčení stacionárního nástroje.

Pozice robotu a jeho pohyby mají vždy vztah k jeho souřadnému systému nástroje, tj. TCP a orientaci nástroje. Abychom získali největší přesnost, je důležité definovat souřadný systém nástroje tak přesně, jak je to možné.

Kalibraci je možné provést také ruční metodou pomocí FlexPendantu (popsáno v *Provozní příručka - IRC5 s FlexPendantem*, sekce *Programování a testování*).

Popis

Abyste mohli definovat TCP a rotaci stacionárního nástroje, potřebujete pohyblivý ukazovací nástroj upevněný na koncovém efektoru robotu.

Před použitím instrukce `SToolRotCalib` musí být splněny některé předpoklady:

- Stacionární nástroj, který se má kalibrovat, musí být upevněn na stacionárně a musí být definován se správným komponentem robhold (`FALSE`).
- Polohovací nástroj (`robhold TRUE`) musí být definován a kalibrován se správnými TCP hodnotami.
- Při používání robotu s absolutní přesností by měla být definována zátěž a těžiště pro polohovací nástroj. `LoadIdentify` se může použít pro definici zátěže.
- Polohovací nástroj, `wobj0`, a`PDispOff` musí být aktivovány před ručním posunem (jogging) robotu.
- Ručně posuňte (jog) TCP polohovacího nástroje co nejblíže k TCP stacionárního nástroje (počátek souřadného systému nástroje) a definujte `robtarget` pro referenční bod `RefTip`
- Ručně posuňte robot (jog) bez změny orientace nástroje tak, aby TCP polohovacího nástroje ukazoval na nějaký bod na kladné z-ose souřadného systému nástroje, a definujte `robtarget` pro `ZPos`bodu.
- Ručně posuňte robot (jog) bez změny orientace nástroje tak, aby TCP polohovacího nástroje ukazoval na nějaký bod na kladné x-ose souřadného systému nástroje, a definujte `robtarget` pro `XPos`bodu.

Jako pomoc pro směrování kladné z-osy a x-osy můžete použít některý typ prodlužovacího nástroje.

Pokračování na další straně

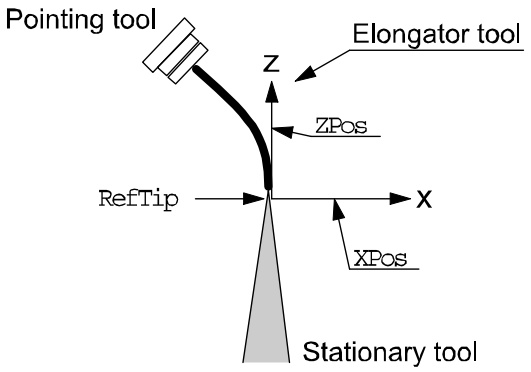
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.261 SToolRotCalib - Kalibrace TCP a rotace pro stacionární nástroj

RobotWare - OS

Pokračování

Definice robtargets `RefTip`, `ZPos`, a `XPos`. Viz obrázek dole.



xx0500002343

POZNÁMKA

Nedoporučuje se modifikovat pozice `RefTip`, `ZPos`, a `XPos` v instrukci `SToolRotCalib`.

Základní příklady

Následující příklad názorně ukazuje instrukci `SToolRotCalib`:

Příklad 1

```
! Created with pointing TCP pointing at the stationary tool
! coordinate system
CONST robtarget pos_tip := [...];
CONST robtarget pos_z := [...];
CONST robtarget pos_x := [...];

PERS tooldata tool1:= [ FALSE, [[0, 0, 0], [1, 0, 0 ,0]], [0, [0,
     0, 0], [1, 0, 0, 0], 0, 0, 0]];

!Instructions for creating or ModPos of pos_tip, pos_z and pos_x
MoveJ pos_tip, v10, fine, point_tool;
MoveJ pos_z, v10, fine, point_tool;
MoveJ pos_x, v10, fine, point_tool;

SToolRotCalib pos_tip, pos_z, pos_x, tool1;
```

Pozice TCP (`tframe.trans`) a orientace nástroje (`tframe.rot`) `tool1` ve světovém souřadném systému jsou vypočítány a aktualizovány.

Argumenty

```
SToolRotCalib RefTip ZPos XPos Tool
```

`RefTip`

Datový typ: `robtarget`

Bod, kde TCP polohovacího nástroje ukazuje na TCP stacionárního nástroje ke kalibraci.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`ZPos`

Datový typ: `robtarget`

Bod prodlužovače, který definuje kladný směr z.

`XPos`

Datový typ: `robtarget`

Bod prodlužovače, který definuje kladný směr x.

`Tool`

Datový typ: `tooldata`

Proměnná perzistentu nástroje, který má být kalibrován.

Vykonávání programu

Systém vypočítává a aktualizuje TCP (`tframe.trans`) a orientaci nástroje (`tfame.rot`) v určeném `tooldata`. Výpočet je založen na určených 3 `robtarget`. Zbývající data v `tooldata` se nemění.

Syntaxe

```
SToolRotCalib
  [ RefTip ':=' ] < expression (IN) of robtarget > ','
  [ ZPos ':=' ] < expression (IN) of robtarget > ','
  [ XPos ':=' ] < expression (IN) of robtarget > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata > ';'
```

Související informace

Pro informace o	Viz
Kalibrace TCP pro pohyblivý nástroj	*MToolTCPCalib - Kalibrace TCP pro pohybující se nástroj na str 437*
Kalibrace rotace pro pohyblivý nástroj	*MToolRotCalib - Kalibrace rotace pro pohybující se nástroj na str 434*
Kalibrace TCP pro stacionární nástroj	*MToolTCPCalib - Kalibrace TCP pro pohybující se nástroj na str 437*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.262 SToolTCPCalib - Kalibrace TCP pro stacionární nástroj

Použití

`SToolTCPCalib` (*(Stationary Tool TCP Calibration)*) se používá pro kalibrování TCP pro stacionární nástroj.

Pozice robotu a jeho pohyby mají vždy vztah k jeho souřadnému systému nástroje, tj. TCP a orientaci nástroje. Abychom získali největší přesnost, je důležité definovat souřadný systém nástroje tak přesně, jak je to možné.

Kalibraci je možné provést také ruční metodou pomocí FlexPendantu (popsáno v *Provozní příručka - IRC5 s FlexPendantem*, sekce *Programování a testování*).

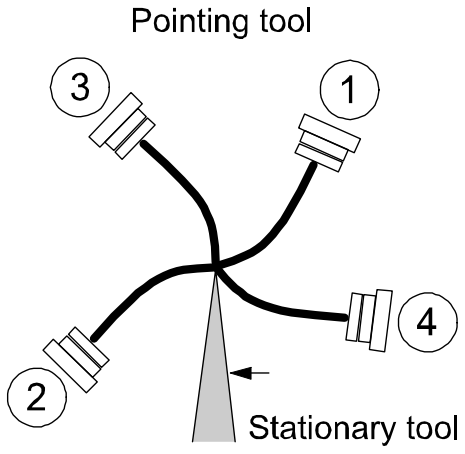
Popis

Abyste mohli definovat TCP stacionárního nástroje, potřebujete pohyblivý polohovací nástroj upevněný na koncovém efektoru robotu.

Následují předpoklady před použitím instrukce `SToolTCPCalib`:

- Stacionární nástroj, který se má kalibrovat, musí být upevněn stacionárně a musí být definován se správným komponentem `robhold` (`FALSE`).
- Polohovací nástroj (`robhold TRUE`) musí být definován a kalibrován se správnými TCP hodnotami.
- Při používání robotu s absolutní přesností by měla být definována zátěž a těžiště pro polohovací nástroj. `LoadIdentify` se může použít pro definici zátěže.
- Polohovací nástroj, `wobj0`, a `PDispOff` musí být aktivovány před ručním posunem (jogging) robotu.
- Ručně posuňte (jog) TCP polohovacího nástroje co nejblíže k TCP stacionárního nástroje a definujte `robtarget` pro první bod p1.
- Definujte další tři pozice p2, p3, a p4,všechny s různými orientacemi.
- Doporučuje se, aby TCP ukazoval různými směry, aby bylo dosaženo spolehlivého statistického výsledku.

Definice 4 joittarget p1...p4, viz obrázek dole.



xx0500002344

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

POZNÁMKA

Nedoporučuje se modifikovat pozice `Pos1` až `Pos4` v instrukci `SToolTCPCalib`.

Reorientace mezi 4 pozicemi by měla být co největší, stavějící robot do odlišných konfigurací. Je také dobrou praxí kontrolovat kvalitu TCP po kalibraci. Což se může provést reorientací nástroje kvůli kontrole, jestli TCP je v klidu.

Základní příklad

Následující příklad názorně ukazuje instrukci `SToolTCPCalib`:

Příklad 1

```
! Created with pointing TCP pointing at the stationary TCP
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];
CONST robtarget p4 := [...];

PERS tooldata tool1:= [ FALSE, [[0, 0, 0], [1, 0, 0 ,0]], [0,001,
     [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0]];
VAR num max_err;
VAR num mean_err;
! Instructions for creating or ModPos of p1 - p4
MoveJ p1, v10, fine, point_tool;
MoveJ p2, v10, fine, point_tool;
MoveJ p3, v10, fine, point_tool;
MoveJ p4, v10, fine, point_tool;

SToolTCPCalib p1, p2, p3, p4, tool1, max_err, mean_err;
```

Hodnota TCP (`tframe.trans`) od `tool1` bude kalibrována a aktualizována. `max_err` a `mean_err` budou uchovávat max. chybu v mm od vypočítaného TCP a střední chybu v mm od vypočítaného TCP.

Argumenty

```
SToolTCPCalib Pos1 Pos2 Pos3 Pos4 Tool MaxErr MeanErr
```

`Pos1`

Datový typ: `robtarget`

Bod prvního přiblížení.

`Pos2`

Datový typ: `robtarget`

Bod druhého přiblížení.

`Pos3`

Datový typ: `robtarget`

Bod třetího přiblížení.

`Pos4`

Datový typ: `robtarget`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.262 SToolTCPCalib - Kalibrace TCP pro stacionární nástroj

RobotWare - OS

Pokračování

Bod čtvrtého přiblížení.

`Tool`

Datový typ: `tooldata`

Proměnná perzistentu nástroje, který má být kalibrován.

`MaxErr`

Datový typ: `num`

Max chyba v mm na jeden bod přiblížení.

`MeanErr`

Datový typ: `num`

Průměrná vzdálenost bodů přiblížení od vypočítaného TCP, tj. jak přesně byl robot polohován ve vztahu ke stacionárnímu TCP.

Vykonávání programu

Systém vypočítává a aktualizuje hodnotu TCP ve světovém souřadném systému (`tframe.trans`) v určeném `tooldata`. Výpočet je založen na určených 4 `robtarget`. Zbývající data v tooldata, jako je orientace nástroje (`tframe.rot`), se nemění.

Syntaxe

```
SToolTCPCalib
  [ Pos1 ':=' ] < expression (IN) of robtarget > ','
  [ Pos2 ':=' ] < expression (IN) of robtarget > ','
  [ Pos3 ':=' ] < expression (IN) of robtarget > ','
  [ Pos4 ':=' ] < expression (IN) of robtarget > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata > ','
  [ MaxErr ':=' ] < variable (VAR) of num > ','
  [ MeanErr' :=' ] < variable (VAR) of num > ';'
```

Související informace

Pro informace o	Viz
Kalibrace TCP pro pohyblivý nástroj	*SToolTCPCalib - Kalibrace TCP pro stacionární nástroj na str 728*
Kalibrace rotace pro pohyblivý nástroj	*MToolRotCalib - Kalibrace rotace pro pohybující se nástroj na str 434*
Kalibrace TCP a rotace pro stacionární nástroj	*SToolRotCalib - Kalibrace TCP a rotace pro stacionární nástroj na str 725*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.263 Stop - Ukončuje vykonávání programu

Použití

`Stop` se používá pro ukončení vykonávání programu. Jakýkoliv pohyb provedený v tomto okamžiku bude dokončen předtím, než je připravena instrukce `Stop`.

Základní příklady

Následující příklad názorně ukazuje instrukci `Stop`:

Viz také *Další příklady na str 733*.

Příklad 1

```
TPWrite "The line to the host computer is broken";
Stop;
```

Vykonávání programu se zastaví po zprávě zapsané na FlexPendant.

Argumenty

```
Stop [ \NoRegain ] | [ \AllMoveTasks ]
```

`[ \NoRegain ]`

Datový typ: `switch`

Určuje další start programu, bez ohledu na to, jestli se postižená mechanická jednotka vrátila do stop pozice nebo nikoliv.

Jestliže je nastaven argument `\NoRegain`, robot a externí osy se nevrátí na stop pozici (jestliže z ní byly ručně odsunuty - jog).

Jestliže je argument vypuštěn a jestliže robot nebo externí osy byly odsunuty ručně (jog) ze stop pozice, potom robot zobrazí dotaz na FlexPendantu. Uživatel může potom odpovědět, jestli by se robot měl vrátit na stop pozici nebo nikoliv.

`[ \AllMoveTasks ]`

Datový typ: `switch`

Určuje, že by měly být zastaveny programy ve všech běžících normálních úlohách vedle aktuální úlohy.

Jestliže argument je vypuštěn, potom bude zastaven pouze ten program v úloze, který vykonává instrukci.

Vykonávání programu

Instrukce zastaví vykonávání programu, když postižené mechanické jednotky v aktuální pohybové úloze dosáhnou nulové rychlosti u pohybu, který je v té době prováděn, a stojí v klidu. Vykonávání programu může být potom restartováno z další instrukce.

Jestliže je instrukce použita bez přepínačů, potom bude postižen pouze program v této úloze.

Jestliže přepínač `AllMoveTasks` je použit v úloze (normální, statická, polostatická), potom se zastaví program v této úloze a všechny normální úlohy. O deklaraci úloh se dozvíte víc v dokumentaci k systémovým parametrům.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.263 Stop - Ukončuje vykonávání programu
RobotWare - OS
Pokračování

Přepínač `NoRegain` je možné používat pouze v pohybových úlohách, jelikož se týká pouze dráhy pohybu.

Jestliže v událostní rutině existuje instrukce `Stop`, potom bude vykonávání rutiny zastaveno a vykonávání pokračuje podle popisu v *`Stop` na str 732*.

Jestliže existuje instrukce `Stop\AllMoveTasks` v událostní rutině v systému `MultiMove`, potom úloha obsahující instrukci pokračuje, jak je popsáno v *`Stop` na str 732*, a všechny další pohybové úlohy vykonávající událostní rutinu pokračují, jak je popsáno v *`Stop \AllMoveTasks` na str 732* (stejný účinek jako u normálního zastavení programu během vykonávání událostní rutiny).

`Stop`

Událostní rutiny	Efekt `Stop` instrukce
`POWER ON`	Vykonávání je zastaveno pro všechny úlohy. Vykonávání nepokračuje v událostní rutině při dalším příkazu startu.
`START`	Vykonávání je zastaveno pro všechny úlohy. Vykonávání pokračuje v událostní rutině při dalším příkazu startu.
`RESTART`	Vykonávání je zastaveno pro všechny úlohy. Vykonávání pokračuje v událostní rutině při dalším příkazu startu.
`STOP`	Vykonávání je zastaveno. Vykonávání nepokračuje v událostní rutině při dalším příkazu startu.
`QSTOP`	Vykonávání je zastaveno. Vykonávání nepokračuje v událostní rutině při dalším příkazu startu.
`RESET`	Vykonávání je zastaveno. Vykonávání nepokračuje v událostní rutině při dalším příkazu startu.

`Stop \AllMoveTasks`

Událostní rutiny	Efekt `Stop \AllMoveTasks` instrukce
`POWER ON`	Vykonávání je zastaveno pro všechny úlohy. Vykonávání nepokračuje v událostní rutině při dalším příkazu startu.
`START`	Vykonávání je zastaveno pro všechny úlohy. Vykonávání pokračuje v událostní rutině při dalším příkazu startu.
`RESTART`	Vykonávání je zastaveno pro všechny úlohy. Vykonávání pokračuje v událostní rutině při dalším příkazu startu.
`STOP`	Vykonávání je zastaveno pro všechny úlohy. Vykonávání nepokračuje v událostní rutině při dalším příkazu startu.
`QSTOP`	Vykonávání je zastaveno pro všechny úlohy. Vykonávání nepokračuje v událostní rutině při dalším příkazu startu.
`RESET`	Vykonávání je zastaveno. Vykonávání nepokračuje v událostní rutině při dalším příkazu startu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Více příkladů jak používat instrukci `Stop` je názorně uvedeno dole.

Příklad 1

```
MoveL p1, v500, fine, tool1;
TPWrite "Jog the robot to the position for pallet corner 1";
Stop \NoRegain;
p1_read := CRobT(\Tool:=tool1 \WObj:=wobj0);
MoveL p2, v500, z50, tool1;
```

Vykonávání programu se zastavuje s robotem na `p1`. Operátor přesune ručně (jog) robot na `p1_read`. Při příštím spuštění programu se robot nevrací na `p1`, takže pozice `p1_read` může být uložena do programu.

Syntaxe

```
Stop
  [ '\' NoRegain ] '|'
  [ '\' AllMoveTasks ]';'
```

Související informace

Pro informace o	Viz
Ukončení provádění programu	*EXIT - Ukončuje vykonávání programu na str 213*
Pouze zastavení pohybů robotu	*StopMove - Zastavuje pohyby robotu na str 736*
Zastavit program pro odlaďování	*Break - Přerušení provádění programu na str 43*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.264 STOpen - Otevřít servo nástroj

Použití

`STOpen` se používá pro otevření servo nástroje.

Základní příklady

Následující příklady názorně ukazují instrukci `STOpen`:

Příklad 1

```
STOpen gun1;
```

Otevřít servo nástroj `gun1`. Čekejte, až se pistole otevře, potom pokračujte s další instrukcí RAPID.

Příklad 2

```
STOpen gun1 \Conc;
```

Otevřít servo nástroj `gun1`. Pokračujte s další instrukcí RAPID bez čekání na otevření pistole.

Příklad 3

```
STOpen "SERVOGUN"\WaitZeroSpeed;
```

Zastavte servo nástroj `SERVOGUN`, čekejte na dokončení každého koordinovaného pohybu a potom otevřete servo nástroj `SERVOGUN`.

Argumenty

```
STOpen ToolName [\WaitZeroSpeed] [\Conc]
```

`ToolName`

Datový typ: `string`

Jméno mechanické jednotky.

`[\WaitZeroSpeed]`

Datový typ: `switch`

Zastavte servo nástroj, čekejte na dokončení každého koordinovaného pohybu a potom otevřete servo nástroj.

`[\Conc]`

Datový typ: `switch`

Následné instrukce se vykonávají během pohybu pistole. Argument se může použít pro zkrácení doby cyklu. To je užitečné, když jsou např. dvě pistole řízeny současně.

Vykonávání programu

Jestliže existuje mechanická jednotka, potom je servo nástroj přikázán k otevření. Síla hrotu je snížena na nulu a rameno nástroje je posunuto zpět na předem uzavřenou pozici (pre_close).

Rameno nástroje je posunuto max rychlostí a zrychlením, jak jsou definovány v systémových parametrech pro odpovídající externí osy. U pohybů ostatních os je rychlost snížena v ručním režimu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Je možné otevřít nástroj během naprogramovaného pohybu robotu, dokud pohyby robotu neobsahují pohyb ramena nástroje. Jestliže nástroj je otevřen během takového pohybu, bude zobrazena chyba `50251 Tool opening failed`. Přepínač `WaitZeroSpeed` je možné použít pro snížení rizika této chyby.

Jestliže se použije přepínač Conc, instrukce bude považována za připravenou před otevřením servo nástroje. Doporučuje se, aby funkce `STIsOpen` byla použita po `STOpen`, aby byly vyloučeny jakékoliv problémy v souběžném režimu.

Více podrobností viz Řízení pohybu servo nástrojů.

Řešení chyb

Jestliže jméno určeného servo nástroje není konfigurovaný servo nástroj, potom je systémová proměnná `ERRNO` nastavena na `ERR_NO_SGUN`.

Jestliže mechanická jednotka servo nástroje není aktivována, potom je systémová proměnná `ERRNO` nastavena na `ERR_SGUN_NOTACT`. Pro aktivaci servo nástroje použijte instrukci `ActUnit`.

Jestliže pozice servo nástroje není iniciována, potom je systémová proměnná `ERRNO` nastavena na `ERR_SGUN_NOTINIT`. Pozice servo nástroje musí být iniciována poprvé, když je instalována pistole nebo po provedení jemné kalibrace. Použijte servisní rutinu `ManServiceCalib` nebo proveďte kalibraci změny hrotu. Opotřebení hrotu bude resetováno.

Jestliže hroty servo nástroje nejsou synchronizovány, potom je systémová proměnná `ERRNO` nastavena na `ERR_SGUN_NOTSYNC`. Hroty servo nástroje musí být synchronizovány, jestliže počítadlo otáček bylo ztraceno a/nebo aktualizováno. Žádná procesní data, jako opotřebení hrotu, nebudou ztracena.

Všechny shora uvedené chyby mohou být zpracovány v chybovém manipulátoru RAPID.

POZNÁMKA

Jestliže instrukce je vyvolána od úlohy v pozadí a došlo k nouzovému zastavení, potom bude instrukce dokončena bez chyby.

Syntaxe

```
STOpen
[ 'ToolName ':=' ] < expression (IN) of string > ',' 
[ '\'WaitZeroSpeed]' ,'
['\'Conc]'
```

Související informace

Pro informace o	Viz
Zavřít servo nástroj	*STClose - Zavřít servo nástroj na str 717*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.265 StopMove - Zastavuje pohyby robotu

Použití

`StopMove` se používá pro dočasné zastavení pohybů robotu a externích os a všech souvisejících procesů. Jestliže instrukce `StartMove` je dána, pohyb a proces se obnoví.

Tato instrukce může být použita, například, v trap rutině k dočasnému zastavení robotu, když se objeví přerušení.

U základnového systému je možné používat tuto instrukci v následujícím typu programových úloh:

- hlavní úloha `T_ROB1` pro zastavení pohybu v této úloze.
- každá další úloha pro zastavení pohybů v hlavní úloze.

U systému MultiMove je možné používat tuto instrukci v následujícím typu programových úloh:

- pohybová úloha pro zastavení pohybu v této úloze.
- nepohybová úloha, pro zastavení pohybu v připojené pohybové úloze. Kromě toho, jestliže pohyb je zastaven v jedné pohybové úloze příslušející ke skupině koordinovaných synchronizovaných úloh, pohyb je zastaven ve všech spolupracujících úlohách.

Základní příklady

Následující příklad názorně ukazuje instrukci `StopMove`:

Viz také *Další příklady na str 737*.

Příklad 1

```
StopMove;
WaitDI ready_input, 1;
StartMove;
```

Pohyb robotu je zastaven, dokud není nastaven vstup `ready_input`.

Argumenty

```
StopMove [\Quick] [\AllMotionTasks]
```

`[\Quick]`

Datový typ: `switch`

Zastavuje robot na dráze tak rychle, jak je to možné.

Bez volitelného parametru `\Quick` se robot zastaví na dráze, ale brzdná vzdálenost je delší (stejná jako u normálního zastavení programu).

`[\AllMotionTasks]`

Datový typ: `switch`

Zastavit pohyb všech mechanických jednotek v systému. Přepínač `[\AllMotionTasks]` může být použit pouze od nepohybové programové úlohy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Pohyby robotu a externích os se zastaví bez zapojení brzd. Všechny procesy spojené s probíhajícím pohybem jsou zastaveny současně se zastavením pohybu.

Vykonávání programu pokračuje po čekání na zastavení robotu a externích os (stojí v klidu).

S přepínačem `\AllMotionTasks` (povoleno pouze z nepohybové programové úlohy) jsou zastaveny pohyby všech mechanických jednotek v systému.

V základnovém systému bez přepínače `\AllMotionTasks` jsou zastaveny pohyby u následujících mechanických jednotek:

- vždy mechanické jednotky v hlavní úloze, nezávisle na tom, která úloha vykonává instrukci `StopMove`.

V systému MultiMove bez přepínače `\AllMotionTasks` jsou zastaveny pohyby u následujících mechanických jednotek:

- mechanické jednotky v pohybové úloze vykonávající `StopMove`.
- mechanické jednotky v pohybové úloze, které jsou připojeny k nepohybové úloze vykonávající `StopMove`. Kromě toho, jestliže mechanické jednotky jsou zastaveny v jedné připojené pohybové úloze příslušející ke skupině koordinovaných synchronizovaných úloh, potom jsou mechanické jednotky zastaveny ve všech spolupracujících úlohách.

Stav `StopMove` v pohybové úloze generovaný od samotné pohybové úlohy bude automaticky nastaven při spuštění této úlohy od začátku.

Stav `StopMove` v připojené pohybové úloze, generovaný od některé nepohybové úlohy, bude automaticky resetován:

- u normální nepohybové úlohy, při spuštění této úlohy od začátku.
- jestliže polostatická nepohybová úloha, při restartu po výpadku napájení, je spouštěna od začátku.
- jestliže statická nepohybová úloha, při installation startu, když je úloha spouštěna od začátku.

Další příklady

Více příkladů instrukce `StopMove` je názorně uvedeno dole.

Příklad 1

```
VAR intnum intno1;
...
PROC main()
  ...
  CONNECT intno1 WITH go_to_home_pos;
  ISignalDI di1,1,intno1;
  ...

TRAP go_to_home_pos
  VAR robtarget p10;
  StopMove;
  StorePath;
  p10:=CRobT(\Tool:=tool1 \WObj:=wobj0);
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.265 StopMove - Zastavuje pohyby robotu
RobotWare - OS
Pokračování

```
  MoveL home,v500,fine,tool1;
  WaitDI di1,0;
  Move L p10,v500,fine,tool1;
  RestoPath;
  StartMove;
ENDTRAP
```

Když je vstup `di1` nastaven na 1, je aktivováno přerušení, které následně aktivuje rutinu přerušení `go_to_home_pos`. Aktuální pohyb je zastaven a robot se namísto toho posune na pozici `home`. Když je `di1` nastaven na 0, robot se vrací na pozici, kde přerušení vzniklo a pokračuje v pohybu podél naprogramované dráhy,

Příklad 2

```
VAR intnum intno1;
...
PROC main()
  ...
  CONNECT intno1 WITH go_to_home_pos;
  ISignalDI di1,1,intno1;
  ...

TRAP go_to_home_pos ()
  VAR robtarget p10;
  StorePath;
  p10:=CRobT(\Tool:=tool1 \WObj:=wobj0);
  MoveL home,v500,fine,tool1;
  WaitDI di1,0;
  MoveL p10,v500,fine,tool1;
  RestoPath;
  StartMove;
ENDTRAP
```

Stejné jako předchozí příklad, ale robot se nepohybuje k výchozí (home) pozici, dokud není aktuální pohybová instrukce dokončena.

Omezení

Pouze jedné z několika nepohybových úloh je povoleno provést ve stejnou dobu sekvenci `StopMove` - `StartMove` proti nějaké pohybové úloze.

Syntaxe

```
StopMove
  ['\'Quick]
  ['\'AllMotionTasks]';'
```

Související informace

Pro informace o	Viz
Pokračování v pohybu	*StartMove - Znovu spouští pohyb robotu na str 707* *StartMoveRetry - Restartuje pohyb robotu a vykonávání na str 710*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Uložit - obnovit dráhu	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742* *RestoPath - Obnovuje cestu po přerušení na str 546*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.266 StopMoveReset - Resetovat pohybový stav zastavení systému

Použití

`StopMoveReset` se používá pro reset pohybového stavu zastavení systému bez spuštění jakýchkoliv pohybů.

Asynchronně pozvednuté chyby pohybů, jako je `ERR_PATH_STOP` nebo konkrétní procesní chyba během pohybů, mohou být ošetřeny v chybovém handleru `ERROR`. Když se taková chyba objeví, pohyby se okamžitě zastaví a pohybový příznak systémového zastavení je nastaven pro aktuální programové úlohy. To znamená, že pohyb není restartován při provádění jakéhokoliv spouštění programu, zatímco ukazatel programu je uvnitř chybového handleru `ERROR`.

Restart pohybů po takové pohybové chybě bude proveden po jedné z těchto činností:

- Provést `StartMove` nebo `StartMoveRetry`.
- Provést `StopMoveReset` a pohyb se obnoví při příštím spuštění programu.

Základní příklady

Následující příklad názorně ukazuje instrukci `StopMoveReset`:

Příklad 1

```
...
ArcL p101, v100, seam1, weld1, weave1, z10, gun1;
...
ERROR
  IF ERRNO=AW_WELD_ERR OR ERRNO=ERR_PATH_STOP THEN
    ! Execute something but without any restart of the movement
    ! ProgStop - ProgStart must be allowed
    ...
    ! No idea to try to recover from this error, so let the error
    ! stop the program
    ...
    ! Reset the move stop flag, so it’s possible to manual restart
    ! the program and the movement after that the program has
    ! stopped
    StopMoveReset;
  ENDIF
ENDPROC
```

Potom, co shora uvedený chybový handler `ERROR` vykonal the `ENDPROC`, vykonávání programu se zastaví a ukazatel je na začátku instrukce `ArcL`. Další spuštění programu restartuje program a pohyb od pozice, kde se objevila původní pohybová chyba.

Argumenty

```
StopMoveReset [\AllMotionTasks]
```

`[\AllMotionTasks]`

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.266 StopMoveReset - Resetovat pohybový stav zastavení systému

RobotWare - OS

Pokračování

Resetovat stav pohybu systémového zastavení u všech mechanických jednotek v systému. Přepínač `[\AllMotionTasks]` může být použit pouze od nepohybové programové úlohy.

Vykonávání programu

Pro restartování aplikace MultiMove v synchronizovaném koordinovaném režimu musí být vykonán `StopMoveReset` ve všech pohybových úlohách, které jsou zapojeny do koordinace.

S přepínačem `\AllMotionTasks` (povoleno pouze z nepohybové programové úlohy) je proveden reset u všech mechanických jednotek v systému.

V základnovém systému bez přepínače `\AllMotionTasks` je reset vždy prováděn u hlavní úlohy, nezávisle na tom, která úloha vykonává instrukci `StopMoveReset`.

U základnového systému je možné používat `StopMoveReset` v následujícím typu programových úloh:

- hlavní úloha `T_ROB1` pro reset stavu zastavení pohybu v této úloze.
- každá další úloha pro reset stavu zastavení pohybů v hlavní úloze.

U systému MultiMove je možné používat tuto instrukci v následujícím typu programových úloh:

- pohybová úloha pro reset stavu zastavení pohybu v této úloze.
- nepohybová úloha, pro reset stavu zastavení pohybu pohybu v připojené pohybové úloze. Kromě toho, jestliže reset stavu zastavení pohybu v jedné připojené pohybové úloze přísluší ke skupině koordinovaných synchronizovaných úloh, stav zastavení pohybu je resetován ve všech spolupracujících úlohách.

Syntaxe

```
StopMoveReset
  ['\'AllMotionTasks]';'
```

Související informace

Pro informace o	Viz
Zastavit pohyb	*StopMove - Zastavuje pohyby robotu na str 736*
Pokračování v pohybu	*StartMove - Znovu spouští pohyb robotu na str 707* *StartMoveRetry - Restartuje pohyb robotu a vykonávání na str 710*
Uložit - obnovit dráhu	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742* *RestoPath - Obnovuje cestu po přerušení na str 546*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.267 StorePath - Uloží dráhu, kde se přerušení objeví

Použití

`StorePath` se používá k uložení vykonávané dráhy pohybu, např. když se objeví chyba nebo přerušení. Chybový handler nebo trap rutina mohou potom spustit nový dočasný pohyb a konečně restartovat původní pohyb, který byl uložen dříve.

Například, tuto instrukci je možné použít k přechodu na servisní pozici nebo k vyčištění pistole, když se objeví chyba.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `StorePath`:

Viz také *Další příklady na str 743*.

Příklad 1

```
StorePath;
```

Aktuální dráha pohybu je uložena pro pozdější použití. Nastavte systém do režimu nezávislého pohybu.

Příklad 2

```
StorePath \KeepSync;
```

Aktuální dráha pohybu je uložena pro pozdější použití. Udržujte režim synchronizovaného pohybu.

Argumenty

```
StorePath [\KeepSync]
```

`[\KeepSync]`

Keep Synchronization

Datový typ: `switch`

Udržuje režim synchronizovaného pohybu po `StorePath \KeepSync`. Přepínač `KeepSync` se může použít pouze když systém je v režimu synchronizovaného pohybu před voláním `StorePath \KeepSync`.

Bez volitelného parametru `\KeepSync` v koordinovaném synchronizovaném systému MultiMove je systém nastaven do režimu nezávislého-polokoordinovaného pohybu. Po vykonání `StorePath` ve všech zapojených úlohách je systém v polokoordinovaném režimu, jestliže se dále použije koordinovaný pracovní objekt. Jinak je v nezávislém režimu. V polokoordinovaném režimu se doporučuje vždy začínat s pohybem v mechanické jednotce, která kontroluje uživatelský rámec před `WaitSyncTask` ve všech zapojených úlohách.

Vykonávání programu

Aktuální dráha pohybu robotu a externích os je uložena. Potom může být spuštěn další pohyb v trap rutině nebo v chybovém handleru. Když je důvod chyby nebo přerušení odstraněn, potom bude uložená dráha pohybu restartována.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Více příkladů jak používat instrukci `StorePath` je názorně uvedeno dole.

Příklad 1

```
TRAP machine_ready
  VAR robtarget p1;
  StorePath;
  p1 := CRobT();
  MoveL p100, v100, fine, tool1;
  ...
  MoveL p1, v100, fine, tool1;
  RestoPath;
  StartMove;
ENDTRAP
```

Když vznikne přerušení, které aktivuje trap rutinu `machine_ready`, dráha pohybu, kterou robot právě vykonává, je zastavena na konci instrukce (ToPoint) a uložena. Potom robot dá do pořádku přerušení, např. výměnou dílu ve stroji. Potom je normální pohyb restartován.

Omezení

Pouze data dráhy pohybu jsou uložena s instrukcí `StorePath`.

Jestliže uživatel chce přikázat pohyby na úrovni nové dráhy, potom musí být uložena aktuální stop pozice přímo po `StorePath` a předtím, než `RestoPath` provede pohyb k uložené stop pozici na dráze.

Pouze jedna dráha pohybu může být uložena současně.

Syntaxe

```
StorePath
  ['\'KeepSync]';'
```

Související informace

Pro informace o	Viz
Obnova dráhy	*RestoPath - Obnovuje cestu po přerušení na str 546*
Další příklady	*RestoPath - Obnovuje cestu po přerušení na str 546* *PathRecStart - Spustit záznamník dráhy na str 467* *SyncMoveResume - Nastavit synchronizované koordinované pohyby na str 764* *SyncMoveSuspend - Nastavit nezávislé-polokoordinované pohyby na str 766*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.268 STTune - Ladění servo nástroje

Použití

`STTune` se používá k ladění/změně parametru servo nástroje. Parametr je změněn dočasně z původní hodnoty, která je nastavena v systémových parametrech. Hodnota nového ladění bude aktivní okamžitě po vykonání instrukce.

`STTune` je užitečné v ladicích procedurách. Ladicí procedura se typicky používá k nalezení optimální hodnoty parametru. Experiment (tj. vykonání programu s pohybem servo nástroje) je opakován s použitím různých hodnot parametrů ladění.

`STTune` by se neměl používat během kalibrace nebo zavírání nástroje.

Základní příklady

Následující příklad názorně ukazuje instrukci `STTune`:

Příklad 1

```
STTune SEOLO_RG, 0.050, CloseTimeAdjust;
```

Parametr servo nástroje `CloseTimeAdjust` je dočasně nastaven na 0,050 sek.

Argumenty

```
STTune MecUnit TuneValue Type
```

`MecUnit`

Datový typ: `mecunit`

Jméno mechanické jednotky.

`TuneValue`

Datový typ: `num`

Nová hodnota ladění.

`Type`

Datový typ: `tunegtype`

Typ parametru. Parametry servo nástroje dostupné pro ladění: `RampTorqRefOpen`, `RampTorqRefClose`, `KV`, `SpeedLimit`, `CollAlarmTorq`, `CollContactPos`, `CollisionSpeed`, `CloseTimeAdjust`, `ForceReadyDelayT`, `PostSyncTime`, `CalibTime`, `CalibForceLow`, `CalibForceHigh`. Tyto typy jsou předdefinovány v systémových parametrech a definují původní hodnoty.

Popis

`RampTorqRefOpen`

Ladí systémový parametr `Ramp when decrease force`, který rozhoduje, jak rychle je uvolněna síla při otevírání nástroje. Jednotkou je Nm/s a typická hodnota je 200.

Odpovídající systémový parametr: předmět *Motion*, typ *Force master*, parametr `ramp_torque_ref_opening`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`RampTorqRefClose`

Ladí systémový parametr `Ramp when increase force`, který rozhoduje, jak rychle je vytvořena síla při otevírání nástroje. Jednotkou je Nm/s a typická hodnota je 80.

Odpovídající systémový parametr: předmět *Motion*, typ *Force master*, parametr `ramp_torque_ref_closing`.

`KV`

Ladí systémový parametr `KV`, který se používá pro omezení rychlosti. Jednotkou je Nms/rad a typickou hodnotou je 1. Více podrobností viz dokumentace k externí ose.

Odpovídající systémový parametr: předmět *Motion*, typ *Force master*, parametr `Kv`.

`SpeedLimit`

Ladí systémový parametr `Speed limit`, který se používá pro omezení rychlosti. Jednotkou je rad/s a typickou hodnotou je 60. Více podrobností viz dokumentace k externí ose.

Odpovídající systémový parametr: předmět *Motion*, typ *Force master*, parametr `speed_limit`.

`CollAlarmTorq`

Ladí systémový parametr `Collision alarm torque`, který se používá pro automatickou kalibraci nových hrotů. Jednotkou je Nm (moment motoru) a typickou hodnotou je 1. Více podrobností viz dokumentace k externí ose.

Odpovídající systémový parametr: předmět *Motion*, typ *Force master*, parametr `alarm_torque`.

`CollContactPos`

Ladí systémový parametr `Collision delta pos`, který se používá pro automatickou kalibraci nových hrotů. Jednotkou je m a typickou hodnotou je 0,002. Více podrobností viz dokumentace k externí ose.

Odpovídající systémový parametr: předmět *Motion*, typ *Force master*, parametr `distance_to_contact_position`.

`CollisionSpeed`

Ladí systémový parametr `Collision speed`, který se používá pro automatickou kalibraci nových hrotů. Jednotkou je m/s a typickou hodnotou je 0,02. Více podrobností viz dokumentace k externí ose.

Odpovídající systémový parametr: předmět *Motion*, typ *Force master*, parametr `col_speed`.

`CloseTimeAdjust`

Nastavení času konstanty, kladné nebo záporné, momentu, kdy hrot nástroje dosáhne kontaktu během zavírání nástroje. Může se používat k mírnému zpoždění zavírání, když se při svařování používá synchronizované předzavírání.

Odpovídající systémový parametr: předmět *Motion*, typ *SG process*, parametr `min_close_time_adjust`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.268 STTune - Ladění servo nástroje
Servo Tool Control
Pokračování

`ForceReadyDelayT`

Prodleva (-y) času konstanty před odesláním signálu Svar připraven po dosažení naprogramované síly.

Odpovídající systémový parametr: předmět *Motion*, typ *SG process*, parametr `pre_sync_delay_time`.

`PostSyncTime`

Očekávání času uvolnění (s) dalšího pohybu robotu po svaru. Tento typ ladění může být naladěn na synchronizaci otevření pistole s příštím pohybem robotu. Synchronizace může selhat, jestliže parametry jsou nastaveny příliš vysoko.

Odpovídající systémový parametr: předmět *Motion*, typ *SG process*, parametr `post_sync_time`.

`CalibTime`

Čas čekání (s) během kalibrace předtím, než je provedena polohovací korekce hrotu nástroje. Nejlepších výsledků docílíte, když nebudete používat příliš nízkou hodnotu jako např. 0,5 sek.

Odpovídající systémový parametr: předmět *Motion*, typ *SG process*, parametr `calib_time`.

`CalibForceLow`

Minimální síla hrotu (N) použitá během kalibrace TipWear. Pro nejlepší výsledek zjištění tloušťky se doporučuje používat minimální naprogramovanou svařovací sílu.

Odpovídající systémový parametr: předmět *Motion*, typ *SG process*, parametr `calib_force_low`.

`CalibForceHigh`

Maximální síla hrotu (N) použitá během kalibrace TipWear. Pro nejlepší výsledek zjištění tloušťky se doporučuje používat maximální naprogramovanou svařovací sílu.

Odpovídající systémový parametr: předmět *Motion*, typ *SG process*, parametr `calib_force_high`.

Vykonávání programu

Určený ladicí typ a ladicí hodnota jsou aktivovány pro určenou mechanickou jednotku. Tato hodnota je použitelná pro všechny pohyby až do naprogramování nové hodnoty pro aktuální mechanickou jednotku nebo až do resetování ladicích typů a hodnot pomocí instrukce `STTuneReset`.

Původní ladicí hodnoty se mohou trvale změnit v systémových parametrech.

Výchozí ladicí hodnoty servo nástrojů se nastavují automaticky

- vykonáním instrukce `STTuneReset`.
- při **Restartu**.

Řešení chyb

Jestliže jméno určeného servo nástroje není konfigurovaný servo nástroj, potom je systémová proměnná `ERRNO` nastavena na `ERR_NO_SGUN`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Chyba může být zpracována v handleru chyb RAPID.

Syntaxe

```
STTune
  [ MecUnit ':=' ] < variable (VAR) of mecunit > ','
  [ TuneValue' :=' ] < expression (IN) of num > ','
  [ 'Type ':='] < expression (IN) of tunegtype > ]';'
```

Související informace

Pro informace o	Viz
Obnovení parametrů servo nástrojů	*TuneReset - Resetování ladění serva na str 885*
Ladění servo nástroje	*Application manual - Additional axes and stand alone controller*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.269 STTuneReset - Resetování ladění servo nástroje

Použití

`STTuneReset` se používá k obnově původních hodnot parametrů servo nástrojů, jestliže byly změněny instrukcí `STTune`.

Základní příklady

Následující příklad názorně ukazuje instrukci `STTuneReset`:

Příklad 1

```
STTuneReset SEOLO_RG;
```

Obnovit *původní hodnoty parametrů servo nástrojů* pro mechanickou jednotku `SEOLO_RG`.

Argumenty

```
STTuneReset MecUnit
```

`MecUnit`

Datový typ: `mecunit`

Jméno mechanické jednotky.

Vykonávání programu

Původní parametry servo nástroje jsou obnoveny.

Toto je také dosaženo v **Restart**.

Řešení chyb

Jestliže jméno určeného servo nástroje není konfigurovaný servo nástroj, potom je systémová proměnná `ERRNO` nastavena na `ERR_NO_SGUN`.

Chyba může být zpracována v handleru chyb RAPID.

Syntaxe

```
STTuneReset
  [ MecUnit ':=' ] < variable (VAR) of mecunit > ','
```

Související informace

Pro informace o	Viz
Ladění parametrů servo nástrojů	*STTune - Ladění servo nástroje na str 744*
Ladění parametrů servo nástrojů	*Application manual - Additional axes and stand alone controller*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.270 SupSyncSensorOff - Zastavit dohled synchronizovaného senzoru

Použití

`SupSyncSensorOff` se používá pro zastavení dohledu nad pohybem robotu a pohybem synchronizovaného senzoru.

Základní příklad

Základní příklad instrukce `SupSyncSensorOff` je názorně uveden dole.

Příklad

```
SupSyncSensorOff SSYNC1;
```

Senzor není nadále pod dohledem.

Argumenty

```
SupSyncSensorOff MechUnit
```

`MechUnit`

Mechanical unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

Syntaxe

```
SupSyncSensorOff
  [ MechUnit ':=' ] < variable (VAR) of mecunit> ';'
```

Související informace

Pro informace o	Viz
Spustit dohled nad synchronizovaným senzorem	*SupSyncSensorOn - Spustit dohled nad synchronizovaným senzorem na str 750*
Sync k senzoru	*SyncToSensor - Synch k senzoru na str 770*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.271 SupSyncSensorOn - Spustit dohled nad synchronizovaným senzorem

Použití

`SupSyncSensorOn` se používá pro spuštění dohledu mezi pohybem robotu a pohybem synchronizovaného senzoru.

Základní příklad

Základní příklad instrukce `SupSyncSensorOn` je názorně uveden dole.

Příklad

```
SupSyncSensorOn Ssync1, 150, 100, 50
```

Mechanická jednotka `Ssync1` je dohlížena, když senzor je polohován mezi 50 a 150. Dohled je ukončen, jestliže vzdálenost mezi robotem a senzorem je kratší než 100.

Argumenty

```
SupSyncSensorOn MechUnit MaxSyncSup SafetyDist MinSyncSup
    [\SafetyDelay]
```

`MechUnit`

Mechanical unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`MaxSyncSup`

Maximal Synchronized supervised position

Datový typ: `num`

Robot bude dohlížet senzor, dokud senzor nepřejede max sync pozici. Když je bod přejet, dohled je zastaven. Jednotkou je mm.

`SafetyDist`

Safety distance

Datový typ: `num`

`Safetydist` je limit rozdílu mezi očekávanou pozicí stroje a skutečnou pozicí stroje. Musí být záporný, tj. model by se měl vždy pohybovat v předstihu před skutečným strojem. V případě klesajících pozic stroje musí být limit záporný v souladu s max rozdílem záporné pozice (a min předstihovou vzdáleností). V případě narůstajících pozic stroje musí být limit kladný v souladu s min rozdílem kladné pozice (a min předstihovou vzdáleností).

Robot spustí alarm, jestliže vzdálenost mezi robotem a senzorem je menší než bezpečnostní vzdálenost. Když je alarm spuštěn, dohled se zastaví.

Jednotkou je mm.

`MinSyncSup`

Minimal synchronized supervised position

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Robot spustí dohled, když senzor je v okně definovaném od pozice `MinSyncSup` k pozici `MaxSyncSup`. Jednotkou je mm.

`[\SafetyDelay]`

Safety delay

Datový typ: `num`

`SafetyDelay` se používá pro nastavení prodlevy mezi naprogramovanou pozicí robotu a dohlíženou pozicí senzoru. Jednotka je v sekundách.

Omezení

Jestliže se používá `SupSynSensorOn` před dokončením instrukce `WaitSensor`, robot se zastaví.

Syntaxe

```
SupSyncSensorOn
  [ MechUnit ':=' ] <variable (VAR) of mecunit> ','
  [ MaxSyncSup ':=' ] < expression (IN) of num > ','
  [ SafetyDist ':=' ] < expression (IN) of num > ','
  [ MinSyncSup ':=' ] < expression (IN) of num >
  [ \SafetyDelay ':=' ] < expression (IN) of num > ';'
```

Související informace

Pro informace o	Viz
Zastavit dohled synchronizovaného senzoru	*SupSyncSensorOff - Zastavit dohled synchronizovaného senzoru na str 749*
Sync k senzoru	*SyncToSensor - Synch k senzoru na str 770*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.272 SyncMoveOff - Ukončit koordinované synchronizované pohyby

Použití

`SyncMoveOff` se používá k ukončení sekvence synchronizovaných pohybů a, ve většině případů, koordinovaných pohybů. Nejprve budou všechny zapojené programové úlohy čekat na synchronizaci ve stop bodu a potom budou plánovače pohybu pro zapojené programové úlohy nastaveny do nezávislého režimu.

Instrukce `SyncMoveOff` se může použít pouze v systému *MultiMove* s doplňkem *Coordinated Robots* a pouze v programových úlohách definovaných jako `Motion Task`.

VAROVÁNÍ

Aby bylo možné dosáhnout bezpečné funkčnosti synchronizace, každý potkávací bod (parametr `SyncID`) musí mít jedinečné jméno. Jméno potkávacího bodu musí být také stejné pro všechny programové úlohy, které by se měly potkat.

Základní příklady

Následující příklad názorně ukazuje instrukci `SyncMoveOff`:

Viz také *Další příklady na str 753*.

Příklad 1

```
!Program example in task T_ROB1

PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;

...
SyncMoveOn sync1, task_list;
...
SyncMoveOff sync2;
...

!Program example in task T_ROB2

PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;

...
SyncMoveOn sync1, task_list;
...
SyncMoveOff sync2;
...
```

Programová úloha, která jako první dosáhne `SyncMoveOff` s identitou `sync2`, čeká, až ostatní úlohy dosáhnou `SyncMoveOff` se stejnou identitou `sync2`. V tomto bodě synchronizace `sync2` jsou plánovače pohybu pro zapojené programové úlohy

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

nastaveny do nezávislého režimu. Potom obě úlohy T_ROB1 a T_ROB2 pokračují ve svém vykonávání.

Argumenty

```
SyncMoveOff SyncID [\TimeOut]
```

SyncID

Synchronization Identity

Datový typ: syncident

Proměnné, které určují jméno bodu odsynchronizace (potkávacího bodu). Datový typ syncident je nehodnotový typ, používá se pouze jako identifikátor pro pojmenování bodu odsynchronizace.

Proměnná musí být definována a musí mít stejné jméno ve všech spolupracujících programových úlohách. Doporučuje se vždy definovat globální proměnnou v každé úloze (VAR syncident ...).

[\TimeOut]

Datový typ: num

Max doba čekání, až jiné programové úlohy dosáhnou bodu odsynchronizace. Tento čas je definován v sekundách (rozlišení 0,001 s).

Jestliže tento čas uběhne předtím, než všechny programové úlohy dosáhnou bodu odsynchronizace, bude volán chybový handler s chybovým kódem ERR_SYNCMOVEOFF. Jestliže neexistuje chybový handler, vykonávání bude zastaveno.

Jestliže tento argument je vypuštěn, programová úloha bude čekat stále.

Vykonávání programu

Programová úloha, která jako první dosáhne SyncMoveOff, čeká, až ostatní určené úlohy dosáhnou SyncMoveOff se stejnou identitou SyncID. V tomto bodě odsynchronizace SyncID je plánovač pohybu pro zapojené programové úlohy nastaven do nezávislého režimu. Potom zapojené programové úlohy pokračují ve svém vykonávání.

Plánovače pohybů pro zapojené programové úlohy jsou nastaveny do nesynchronizovaného režimu. To znamená následující:

- Všechny programové úlohy RAPID a všechny pohyby od těchto úloh pracují znovu nezávisle na sobě.
- Žádná pohybová instrukce **nesmí** být označena žádným ID číslem. Viz instrukce MoveL.

Je možné vyloučit programové úlohy z důvodů testování z FlexPendantu - Panel volby úkolů. Instrukce SyncMoveOn a SyncMoveOff budou stále pracovat se sníženým počtem programových úloh, i pro pouze jednu programovou úlohu.

Další příklady

Více příkladů jak používat instrukci SyncMoveOff je názorně uvedeno dole.

Příklad jednoduchého synchronizovaného pohybu

```
!Program example in task T_ROB1
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.272 SyncMoveOff - Ukončit koordinované synchronizované pohyby

RW-MRS Synchronized

Pokračování

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;

PROC main()
  ...
  MoveL p_zone, vmax, z50, tcp1;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, tcp1;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, tcp1 \WObj:= rob2_obj;
  MoveL * \ID:=20, v100, fine, tcp1 \WObj:= rob2_obj;
  SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
ENDPROC

!Program example in task T_ROB2
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;

PROC main()
  ...
  MoveL p_zone, vmax, z50, obj2;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, obj2;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, obj2;
  MoveL * \ID:=20, v100, fine, obj2 ;
  SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
ENDPROC
```

Nejprve programové úlohy `T_ROB1` a `T_ROB2` čekají na sebe u `WaitSyncTask` s identitou `sync1`, naprogramovány s rohovou dráhou pro předchozí pohyby kvůli úspoře doby cyklu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Potom programové úlohy čekají na sebe u SyncMoveOn s identitou sync2, naprogramovány s nezbytným stop bodem pro předchozí pohyby. Potom je plánovač pohybů pro zapojené programové úlohy nastaven do synchronizovaného režimu.

Potom T_ROB2 přesouvá obj2 do ID bodu10 a 20 ve světovém souřadném systému, zatímco T_ROB1 přesouvá tcp1 do ID bodu10 a 20 na pohybujícím se objektu obj2.

Potom programové úlohy čekají na sebe u SyncMoveOff s identitou sync3, naprogramovány s nezbytným stop bodem pro předchozí pohyby. Potom je plánovač pohybů pro zapojené programové úlohy nastaven do nezávislého režimu.

Příklad obnovy po chybě

```
!Program example with use of time-out function
VAR syncident sync3;

...
SyncMoveOff sync3 \TimeOut := 60;
...
ERROR
  IF ERRNO = ERR_SYNCMOVEOFF THEN
    RETRY;
  ENDIF
```

Programová úloha čeká na instrukci SyncMoveOff a až některá jiná programová úloha dosáhne stejného bodu synchronizace sync3. Po čekání 60 sekund je volán chybový handler s ERRNO totožnou s ERR_SYNCMOVEOFF. Potom je instrukce SyncMoveOff volána znovu, aby čekala dalších 60 sekund.

Příklad s polokoordinovaným a koordinovaným pohybem

```
!Example with semicoordinated and synchronized movement
!Program example in task T_ROB1
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
PERS wobjdata rob2_obj:= [FALSE,FALSE,"ROB_2",
     [[0,0,0],[1,0,0,0]],[[155.241,-51.5938,57.6297],
     [0.493981,0.506191,-0.501597,0.49815]]];
VAR syncident sync0;
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
VAR syncident sync4;

PROC main()
  ...
  WaitSyncTask sync0, task_list;
  MoveL p1_90, v100, fine, tcp1 \WObj:= rob2_obj;
  WaitSyncTask sync1, task_list;
  SyncMoveOn sync2, task_list;
  MoveL p1_100 \ID:=10, v100, fine, tcp1 \WObj:= rob2_obj;
  SyncMoveOff sync3;
  !Wait until the movement has been finished in T_ROB2
  WaitSyncTask sync3, task_list;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.272 SyncMoveOff - Ukončit koordinované synchronizované pohyby

RW-MRS Synchronized

Pokračování

```
  !Now a semicoordinated movement can be performed
  MoveL p1_120, v100, z10, tcp1 \WObj:= rob2_obj;
  MoveL p1_130, v100, fine, tcp1 \WObj:= rob2_obj;
  WaitSyncTask sync4, task_list;
  ...
ENDPROC

!Program example in task T_ROB2
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync0;
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
VAR syncident sync4;

PROC main()
  ...
  MoveL p_fine, v1000, fine, tcp2;
  WaitSyncTask sync0, task_list;
  !Wait until the movement in T_ROB1 task is finished
  WaitSyncTask sync1, task_list;
  SyncMoveOn sync2, task_list;
  MoveL p2_100 \ID:=10, v100, fine, tcp2;
  SyncMoveOff sync3;
  !The path has been removed at SyncMoveOff
  !Perform a movement to wanted position for the object to make
       the position available for other tasks
  MoveL p2_100, v100, fine, tcp2;
  WaitSyncTask sync3, task_list;
  WaitSyncTask sync4, task_list;
  MoveL p2_110, v100, z10, tcp2;
  ...
ENDPROC
```

Při přepínání mezi polokoordinovaným a koordinovaným pohybem je nutný `WaitSyncTask` **(při používání identity** `sync1`**).**

Při přepínání mezi synchronizovaným a polosynchronizovaným pohybem se potřebuje úloha, která pohybuje pracovním objektem (`rob2_obj`**), přemístit do požadované pozice. Potom je třeba** `WaitSyncTask` **(identita** `sync3`**), než může být proveden polokoordinovaný pohyb.**

Řešení chyb

Jestliže je dosaženo časového limitu, protože `SyncMoveOff` **není připraven včas, potom je systémová proměnná** `ERRNO` **nastavena na** `ERR_SYNCMOVEOFF`**.**

Tato chyba může být zpracována v chybovém handleru `ERROR`**.**

Omezení

Instrukce `SyncMoveOff` **může být vykonána pouze v případě, že všechny zapojené roboty stojí v klidu ve stop bodu.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (zonedata `fine`), nikoliv s fly-by bodem, jinak nebude možný restart po selhání napájení.

`SyncMoveOff` se nemůže provádět v RAPID rutině připojené k jakékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart, Reset nebo Step.

Syntaxe

```
SyncMoveOff
  [ SyncID ':=' ] < variable (VAR) of syncident>
  [ '\'TimeOut' :=' < expression (IN) of num> ] ';'
```

Související informace

Pro informace o	Viz
Určit spolupracující programové úlohy	*tasks - Programové úlohy RAPID na str 1607*
Identita pro bod synchronizace	*syncident - Identita pro bod synchronizace na str 1603*
Spustit koordinované synchronizované pohyby	*SyncMoveOn - Spustit koordinované synchronizované pohyby na str 758*
Nastavit nezávislé pohyby	*SyncMoveUndo - Nastavit nezávislé pohyby na str 768*
Testovat synchronizovaný režim	*IsSyncMoveOn - Testovat synchronizovaný režim na str 1218*
Systém MultiMove s doplňkem Coordinated robots	*Application manual - MultiMove*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.273 SyncMoveOn - Spustit koordinované synchronizované pohyby

Použití

`SyncMoveOn` se používá ke spuštění sekvence synchronizovaných pohybů a, ve většině případů, koordinovaných pohybů. Nejprve budou všechny zapojené programové úlohy čekat na synchronizaci ve stop bodu a potom bude plánovač pohybu pro zapojené programové úlohy nastaven do synchronizovaného režimu.

Instrukce `SyncMoveOn` se může použít pouze v systému *MultiMove* s doplňkem *Coordinated Robots* a pouze v programových úlohách definovaných jako `Motion Task`.

VAROVÁNÍ

Aby bylo možné dosáhnout bezpečné funkčnosti synchronizace, každý potkávací bod (parametr `SyncID`) musí mít jedinečné jméno. Jméno potkávacího bodu musí být také stejné pro všechny programové úlohy, které by se měly potkat v potkávacím bodě.

Základní příklady

Následující příklad názorně ukazuje instrukci `SyncMoveOn`:

Viz také *Další příklady na str 760*.

Příklad 1

```
!Program example in task T_ROB1

PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;

...
SyncMoveOn sync1, task_list;
...
SyncMoveOff sync2;
...

!Program example in task T_ROB2

PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;

...
SyncMoveOn sync1, task_list;
...
SyncMoveOff sync2;
...
```

Programová úloha, která jako první dosáhne `SyncMoveOn` s identitou `sync1`, čeká, až jiná úloha dosáhne svého `SyncMoveOn` se stejnou identitou `sync1`. V tomto

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

bodě synchronizace `sync1` je plánovač pohybu pro zapojené programové úlohy nastaven do synchronizovaného režimu. Potom obě úlohy `T_ROB1` a `T_ROB2` pokračují ve svém vykonávání, a jsou synchronizovány, dokud nedosáhnou `SyncMoveOff` se stejnou identitou `sync2`.

Argumenty

```
SyncMoveOn SyncID TaskList [\TimeOut]
```

`SyncID`

Synchronization Identity

Datový typ:`syncident`

Proměnná, která určuje jméno bodu synchronizace (potkávacího bodu). Datový typ `syncident` je nehodnotový typ, používá se pouze jako identifikátor pro pojmenování bodu synchronizace.

Proměnná musí být definována a musí mít stejné jméno ve všech spolupracujících programových úlohách. Doporučuje se vždy definovat globální proměnnou v každé úloze (`VAR syncident ...`).

`TaskList`

Datový typ:`tasks`

Perzistentní proměnná, která v seznamu úloh (pole) určuje jméno (`string`) programových úloh, které by se měly potkat v bodě synchronizace se jménem podle argumentu `SyncID`.

Perzistentní proměnná musí být definována a musí mít stejné jméno a stejný obsah ve všech spolupracujících programových úlohách. Doporučuje se vždy definovat globální proměnnou v systému (`PERS tasks ...`).

`[\TimeOut]`

Datový typ: `num`

Max doba čekání, až jiné programové úlohy dosáhnou bodu synchronizace. Tento čas je definován v sekundách (rozlišení 0,001 s).

Jestliže tento čas uběhne předtím, než všechny programové úlohy dosáhnou bodu synchronizace, bude volán chybový handler s chybovým kódem `ERR_SYNCMOVEON`. Jestliže neexistuje chybový handler, vykonávání bude zastaveno.

Jestliže tento argument je vypuštěn, programová úloha bude čekat stále.

Vykonávání programu

Programová úloha, která jako první dosáhne `SyncMoveOn`, čeká, až ostatní určené úlohy dosáhnou svého `SyncMoveOn` se stejnou identitou `SyncID`. V tomto bodě synchronizace `SyncID` je plánovač pohybu pro zapojené programové úlohy nastaven do synchronizovaného režimu. Potom zapojené programové úlohy pokračují ve svém vykonávání.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.273 SyncMoveOn - Spustit koordinované synchronizované pohyby
RW-MRS Independent
Pokračování

Plánovač pohybů pro zapojené programové úlohy jsou nastaven do synchronizovaného režimu. To znamená následující:

- Každá pohybová instrukce v každé programové úloze v `TaskList` pracuje synchronně s pohybovými instrukcemi v ostatních programových úlohách v `TaskList`.
- Všechny spolupracující pohybové instrukce jsou naplánovány a interpolovány ve stejném plánovači pohybů.
- Všechny pohyby začínají a končí ve stejnou dobu. Pohyb, který trvá nejdelší dobu, bude rychlostním vzorem se sníženou rychlostí ve vztahu k pracovnímu objektu pro jiné pohyby.
- Všechny spolupracující pohybové instrukce musí být označeny stejným ID číslem. Viz instrukce `MoveL`.

Je možné vyloučit programové úlohy z důvodů testování z FlexPendantu - Panel volby úkolů. Instrukce `SyncMoveOn` bude stále pracovat se sníženým počtem programových úloh, i pro pouze jednu programovou úlohu.

Další příklady

Více příkladů jak používat instrukci `SyncMoveOn` je názorně uvedeno dole.

Příklad 1

```
!Program example in task T_ROB1
PERS tasks task_list{2} := [["T_ROB1"], ["T_ROB2"]];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;

PROC main()
  ...
  MoveL p_zone, vmax, z50, tcp1;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, tcp1;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, tcp1 \WOBJ:= rob2_obj;
  MoveL * \ID:=20, v100, fine, tcp1 \WOBJ:= rob2_obj;
  SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
ENDPROC
!Program example in task T_ROB2
PERS tasks task_list{2} := [["T_ROB1"], ["T_ROB2"]];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
PROC main()
  ...
  MoveL p_zone, vmax, z50, obj2;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, obj2;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, obj2;
  MoveL * \ID:=20, v100, fine, obj2;
  SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
ENDPROC
```

Nejprve programové úlohy T_ROB1 **a** T_ROB2 **čekají na sebe u** WaitSyncTask **s identitou** sync1**. Jsou naprogramovány s rohovou dráhou pro předchozí pohyby kvůli úspoře doby cyklu.**

Potom programové úlohy čekají na sebe u SyncMoveOn **s identitou** sync2**. Jsou naprogramovány s nezbytným stop bodem pro předchozí pohyby. Potom je plánovač pohybů pro zapojené programové úlohy nastaven do synchronizovaného režimu.**

Potom T_ROB2 **přesouvá** obj2 **do** ID **bodu**10 **a** 20 **ve světovém souřadném systému, zatímco** T_ROB1 **přesouvá** tcp1 **do** ID **bodu**10 **a** 20 **na pohybujícím se objektu** obj2**.**

Příklad 2

```
!Program example with use of time-out function
VAR syncident sync3;

...
SyncMoveOn sync3, task_list \TimeOut :=60;
...
ERROR
  IF ERRNO = ERR_SYNCMOVEON THEN
    RETRY;
  ENDIF
```

Programová úloha čeká na instrukci SyncMoveOn**, aby programová úloha** T_ROB2 **dosáhla stejného bodu synchronizace** sync3**. Po čekání 60 sekund je volán chybový handler s** ERRNO **totožnou s** ERR_SYNCMOVEON**. Potom je instrukce** SyncMoveOn **volána znovu, aby čekala dalších 60 sekund.**

Příklad 3 - Příklad programu se třemi úlohami

```
!Program example in task T_ROB1
PERS tasks task_list1 {2} :=[["T_ROB1"], ["T_ROB2"]];
PERS tasks task_list2 {3} :=[["T_ROB1"], ["T_ROB2"], ["T_ROB3"]];
VAR syncident sync1;
...
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.273 SyncMoveOn - Spustit koordinované synchronizované pohyby
RW-MRS Independent
Pokračování

```
VAR syncident sync5;

...
  SyncMoveOn sync1, task_list1;
  ...
  SyncMoveOff sync2;
  WaitSyncTask sync3, task_list2;
  SyncMoveOn sync4, task_list2;
  ...
  SyncMoveOff sync5;
...

!Program example in task T_ROB2

PERS tasks task_list1 {2} := [["T_ROB1"], ["T_ROB2"]];
PERS tasks task_list2 {3} := [["T_ROB1"], ["T_ROB2"], ["T_ROB3"]];
VAR syncident sync1;
...
VAR syncident sync5;

...
  SyncMoveOn sync1, task_list1;
  ...
  SyncMoveOff sync2;
  WaitSyncTask sync3, task_list2;
  SyncMoveOn sync4, task_list2;
  ...
  SyncMoveOff sync5;
...

!Program example in task T_ROB3

PERS tasks task_list2 {3} := [["T_ROB1"], ["T_ROB2"], ["T_ROB3"]];
VAR syncident sync3;
VAR syncident sync4;
VAR syncident sync5;

...
  WaitSyncTask sync3, task_list2;
  SyncMoveOn sync4, task_list2;
  ...
  SyncMoveOff sync5;
...
```

V tomto příkladu se nejprve programové úlohy `T_ROB1` a `T_ROB2` pohybují synchronizovaně a `T_ROB3` se pohybuje nezávisle. Dále v programu se všechny tři úlohy pohybují synchronizovaně. Aby se zabránilo vykonání instrukce `SyncMoveOn` v `T_ROB3` před ukončením první synchronizace `T_ROB1` a `T_ROB2`, použije se instrukce `WaitSyncTask`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Jestliže je dosaženo časového limitu, protože `SyncMoveOn` není připraven včas, potom je systémová proměnná `ERRNO` nastavena na `ERR_SYNCMOVEON`.

Tato chyba může být zpracována v chybovém handleru `ERROR`.

Omezení

Instrukce `SyncMoveOn` může být vykonána pouze v případě, že všechny zapojené roboty stojí v klidu ve stop bodu.

Pouze jedna skupina koordinovaných synchronizovaných pohybů může být aktivní ve stejné době.

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (zonedata `fine`), nikoliv s fly-by bodem, jinak nebude možný restart po selhání napájení.

`SyncMoveOn` se nemůže provádět v RAPID rutině připojené k jakékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart, Reset nebo Step.

Syntaxe

```
SyncMoveOn
  [ SyncID ':=' ] < variable (VAR) of syncident> ','
  [ TaskList ':=' ] < persistent array {*} (PERS) of tasks> ','
  [ '\' TimeOut ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Určit spolupracující programové úlohy	*tasks - Programové úlohy RAPID na str 1607*
Identita pro bod synchronizace	*syncident - Identita pro bod synchronizace na str 1603*
Ukončit koordinované synchronizované pohyby	*SyncMoveOff - Ukončit koordinované synchronizované pohyby na str 752*
Nastavit nezávislé pohyby	*SyncMoveUndo - Nastavit nezávislé pohyby na str 768*
Testovat synchronizovaný režim	*IsSyncMoveOn - Testovat synchronizovaný režim na str 1218*
Systém MultiMove s doplňkem Coordinated Robots	*Application manual - MultiMove*
Čekat na synchronizované úlohy	*WaitSyncTask - Čekat v bodu synchronizace na jiné programové úlohy na str 956*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.274 SyncMoveResume - Nastavit synchronizované koordinované pohyby

Použití

SyncMoveResume se používá pro přechod zpět k synchronizovaným pohybům z režimu nezávislých pohybů. Instrukci je možné použít pouze na úrovni StorePath, např. po vykonání StorePath \KeepSync, když systém je v režimu nezávislých pohybů po vykonání SyncMoveSuspend. Aby bylo možné instrukci použít, systém musí být v režimu synchronizovaných pohybů před vykonáním instrukce StorePath a SyncMoveSuspend.

Instrukce SyncMoveResume se může použít pouze v systému *MultiMove* s doplňky *Coordinated Robots* a *Path Recovery* a pouze v programových úlohách definovaných jako *Motion Task*.

Základní příklady

Následující příklad názorně ukazuje instrukci SyncMoveResume:

Příklad 1

```
ERROR
  StorePath \KeepSync;
  ! Save position
  p11 := CRobT(\Tool:=tool2);
  ! Move in synchronized motion mode
  MoveL p12\ID:=111, v50, fine, tool2;
  SyncMoveSuspend;
  ! Move in independent mode somewhere, e.g. to a cleaning station
  p13 := CRobT();
  MoveL p14, v100, fine, tool2;
  ! Do something at cleaning station
  MoveL p13, v100, fine, tool2;
  SyncMoveResume;
  ! Move in synchronized motion mode back to start position p11
  MoveL p11\ID:=111, fine, z20, tool2;
  RestoPath;
  StartMove;
  RETRY;
```

Objevuje se některý druh řešitelné chyby. Systém je udržován v synchronizovaném režimu a synchronizovaný pohyb je proveden k bodu, např. pohybem zpět na dráze. Potom je nezávislý pohyb proveden k čisticí stanici. Potom je robot posunut zpět k bodu, kde se objevila chyba a program pokračuje, kde byl přerušen chybou.

Vykonávání programu

SyncMoveResume si vynucuje obnovení synchronizovaného režimu, když systém je v režimu nezávislých pohybů na úrovni StorePath.

SyncMoveResume se vyžaduje ve všech úlohách, které byly vykonávány v synchronizovaném pohybu před vstoupením do režimu nezávislých pohybů. Jestliže jedna pohybová úloha vykonává SyncMoveResume, potom tato úloha bude čekat, až všechny úlohy, které byly dříve v režimu synchronizovaných pohybů vykonají

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

instrukci `SyncMoveResume` . Potom zapojené programové úlohy pokračují ve svém vykonávání.

Omezení

`SyncMoveResume` se může používat pouze k přechodu do režimu synchronizovaných pohybů a může být použita pouze na úrovni `StorePath`.

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (zonedata `fine`), nikoliv s fly-by bodem, jinak nebude možný restart po selhání napájení.

`SyncMoveResume` se nemůže provádět v RAPID rutině připojené k jakékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart,, Reset nebo Step.

Syntaxe

```
SyncMoveResume ';'
```

Související informace

Pro informace o	Viz
Určit spolupracující programové úlohy	*tasks - Programové úlohy RAPID na str 1607*
Spustit koordinované synchronizované pohyby	*SyncMoveOn - Spustit koordinované synchronizované pohyby na str 758*
Ukončit koordinované synchronizované pohyby	*SyncMoveOff - Ukončit koordinované synchronizované pohyby na str 752*
Testovat synchronizovaný režim	*SyncMoveOn - Spustit koordinované synchronizované pohyby na str 758*
Ukládá dráhu	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742*
Obnovuje dráhu	*RestoPath - Obnovuje cestu po přerušení na str 546*
Pozastavuje synchronizované pohyby	*SyncMoveSuspend - Nastavit nezávislé-polokoordinované pohyby na str 766*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.275 SyncMoveSuspend - Nastavit nezávislé-polokoordinované pohyby

Použití

SyncMoveSuspend se používá k pozastavení režimu synchronizovaných pohybů a nastavení systému na režim nezávislých-polokoordinovaných pohybů. Instrukci je možné používat pouze na úrovni StorePath, např. po vykonání StorePath nebo StorePath \KeepSync, když je systém v režimu synchronizovaných pohybů.

Instrukce SyncMoveSuspend se může použít pouze v systému *MultiMove System* s doplňky *Coordinated Robots* a *Path Recovery* a pouze v programových úlohách definovaných jako *Motion Task*.

Základní příklady

Následující příklad názorně ukazuje instrukci SyncMoveSuspend:

Příklad 1

```
ERROR
  StorePath \KeepSync;
  ! Save position
  p11 := CRobT(\Tool:=tool2);
  ! Move in synchronized motion mode
  MoveL p12\ID:=111, v50, fine, tool2;
  SyncMoveSuspend;
  ! Move in independent mode somewhere, e.g. to a cleaning station
  p13 := CRobT();
  MoveL p14, v100, fine, tool2;
  ! Do something at cleaning station
  MoveL p13, v100, fine, tool2;
  SyncMoveResume;
  ! Move in synchronized motion mode back to start position p11
  MoveL p11\ID:=111, fine, z20, tool2;
  RestoPath;
  StartMove;
  RETRY;
```

Objevuje se některý druh řešitelné chyby. Systém je udržován v synchronizovaném režimu a synchronizovaný pohyb je proveden k bodu, např. pohybem zpět na dráze. Potom je nezávislý pohyb proveden k čisticí stanici. Potom je robot posunut zpět k bodu, kde se objevila chyba a program pokračuje, kde byl přerušen chybou.

Vykonávání programu

SyncMoveSuspend vynucuje reset synchronizovaných pohybů a nastavuje systém do režimu nezávislých-polokoordinovaných pohybů.

SyncMoveSuspend se vyžaduje ve všech úlohách se synchronizovaným pohybem pro nastavení systému do režimu nezávislých-polokoordinovaných pohybů. Jestliže jedna pohybová úloha vykonává SyncMoveSuspend, potom tato úloha čeká, až ostatní úlohy vykonají instrukci SyncMoveSuspend.

Po vykonání SyncMoveSuspend ve všech zapojených úlohách je systém v polokoordinovaném režimu, jestliže dále používá koordinovaný pracovní objekt. Jinak je v nezávislém režimu. V polokoordinovaném režimu se doporučuje vždy

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

začínat s pohybem v mechanické jednotce, která kontroluje uživatelský rámec před `WaitSyncTask` ve všech zapojených úlohách.

Omezení

Instrukce `SyncMoveSuspend` pozastavuje synchronizovaný režim pouze na úrovni `StorePath`. Po návratu z úrovně `StorePath` je systém nastaven do režimu, ve kterém byl před `StorePath`.

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (zonedata `fine`), nikoliv s fly-by bodem, jinak nebude možný restart po selhání napájení.

`SyncMoveSuspend` se nemůže provádět v RAPID rutině připojené k jakékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart,, Reset nebo Step.

Syntaxe

```
SyncMoveSuspend ';'
```

Související informace

Pro informace o	Viz
Určit spolupracující programové úlohy	*tasks - Programové úlohy RAPID na str 1607*
Spustit koordinované synchronizované pohyby	*SyncMoveOn - Spustit koordinované synchronizované pohyby na str 758*
Ukončit koordinované synchronizované pohyby	*SyncMoveOff - Ukončit koordinované synchronizované pohyby na str 752*
Testovat synchronizovaný režim	*IsSyncMoveOn - Testovat synchronizovaný režim na str 1218*
Ukládá dráhu	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742*
Obnovuje dráhu	*RestoPath - Obnovuje cestu po přerušení na str 546*
Obnovuje synchronizované pohyby	*SyncMoveResume - Nastavit synchronizované koordinované pohyby na str 764*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.276 SyncMoveUndo - Nastavit nezávislé pohyby

Použití

`SyncMoveUndo` vynucuje reset synchronizovaných koordinovaných pohybů a nastavuje systém do režimu nezávislých pohybů.

Instrukce `SyncMoveUndo` se může použít pouze v systému *MultiMove* s doplňkem *Coordinated Robots* a pouze v programových úlohách definovaných jako `Motion Task`.

Základní příklady

Následující příklad názorně ukazuje instrukci `SyncMoveUndo`:

Příklad 1

Příklad programu v úloze `T_ROB1`

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
PROC main()

...
  MoveL p_zone, vmax, z50, tcp1;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, tcp1;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, tcp1 \WOBJ:= rob2_obj;
  MoveL * \ID:=20, v100, fine, tcp1 \WOBJ:= rob2_obj;
  SyncMoveOff sync3;
UNDO
  SyncMoveUndo;
ENDPROC
```

Jestliže program je zastaven, když vykonávání probíhá uvnitř procedury `syncmove` a ukazatel programu je posunut mimo proceduru `syncmove`, potom jsou vykonány všechny instrukce uvnitř `UNDO` handleru. V tomto příkladu je vykonána instrukce `SyncMoveUndo` a systém je nastaven do režimu nezávislých pohybů.

Vykonávání programu

Vynutit reset synchronizovaných koordinovaných pohybů a nastavit systém do režimu nezávislých pohybů.

Stačí vykonat `SyncMoveUndo` v jedné programové úloze, aby se celý systém nastavil do režimu nezávislých pohybů. Instrukci je možné vykonat několikrát bez chyby, jestliže systém je již v režimu nezávislých pohybů.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Systém je nastaven také do výchozího režimu nezávislých pohybů

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Syntaxe

```
SyncMoveUndo ';'
```

Související informace

Pro informace o	Viz
Určit spolupracující programové úlohy	*tasks - Programové úlohy RAPID na str 1607*
Identita pro bod synchronizace	*syncident - Identita pro bod synchronizace na str 1603*
Spustit koordinované synchronizované pohyby	*SyncMoveOn - Spustit koordinované synchronizované pohyby na str 758*
Ukončit koordinované synchronizované pohyby	*SyncMoveOff - Ukončit koordinované synchronizované pohyby na str 752*
Testovat synchronizovaný režim	*IsSyncMoveOn - Testovat synchronizovaný režim na str 1218*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.277 SyncToSensor - Synch k senzoru

Použití

`SyncToSensor` se používá ke spuštění nebo zastavení synchronizace pohybu robotu k pohybu senzoru.

Základní příklady

Základní příklady instrukce `SyncToSensor` jsou názorně uvedeny dole.

Příklad 1

```
WaitSensor Ssync1;
MoveL *, v1000, z10, tool, \WObj:=wobj0;
SyncToSensor Ssync1\On;
MoveL *, v1000, z20, tool, \WObj:=wobj0;
MoveL *, v1000, z20, tool, \WObj:=wobj0;
SyncToSensor Ssync1\Off;
```

Argumenty

```
SyncToSensor MechUnit [\MaxSync] [\On] | [\Off]
```

`MechUnit`

Mechanical Unit

Datový typ: `mecunit`

Pohybující se mechanická jednotka, ke které se vztahuje pozice robotu v instrukci.

`[\MaxSync]`

Datový typ: `num`

Robot se bude bude pohybovat synchronizovaně se senzorem, dokud senzor nepřejde pozici `MaxSync`. Potom se robot bude pohybovat nesynchronizovaně naprogramovanou rychlostí. Jestliže volitelný parametr `MaxSync` není definován, robot se bude pohybovat synchronizovaně až do vykonání instrukce `SyncToSensor Ssync1\Off`.

`[\On]`

Datový typ: `switch`

Robot se pohybuje synchronizovaně se senzorem po instrukci pomocí argumentu `\On`.

`[\Off]`

Datový typ: `switch`

Robot se pohybuje nesynchronizovaně se senzorem po instrukci pomocí argumentu `\Off`.

Vykonávání programu

`SyncToSensor Ssync1 \On` znamená, že robot se začíná pohybovat synchronizovaně se senzorem `Ssync1`. Takže robot přechází na naprogramovaný `robtarget` ve stejné době jako senzor přechází externí pozici uloženou v `robtarget`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`SyncToSensor Ssync1 \Off` **znamená, že robot zastavuje pohyb synchronizovaně se senzorem.**

Omezení

Jestliže instrukce `SyncToSensor Ssync1 \On` **je vydána, zatímco senzor nebyl připojen přes** `WaitSensor`**, potom se robot zastaví.**

Syntaxe

```
SyncToSensor
  [ MechUnit ':=' ] < variable (VAR) of mecunit >
  [ \MaxSync] [ '\' On] | [ '\' Off] ';'
```

Související informace

Pro informace o	Viz
Spustit dohled nad synchronizovaným senzorem	*SupSyncSensorOn - Spustit dohled nad synchronizovaným senzorem na str 750*
Sync k senzoru	*SyncToSensor - Synch k senzoru na str 770*
Čekejte na připojení na senzor	*WaitSensor - Čekat na připojení na senzor na str 953*
Shodit objekt na senzor	*DropSensor - Shodit objekt nebo senzor na str 157*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.278 SystemStopAction - Zastavit systém robotu

Použití

`SystemStopAction` se může použít k zastavení systému robotu různými způsoby podle závažnosti chyby nebo problému.

Základní příklady

Následující příklady názorně ukazují instrukci `SystemStopAction`:

Příklad 1

```
SystemStopAction \Stop;
```

Tím se zastaví vykonávání programu a pohyby robotu ve všech pohybových úlohách. Žádná konkrétní činnost není nutná před restartováním vykonávání programu.

Příklad 2

```
SystemStopAction \StopBlock;
```

Tím se zastaví vykonávání programu a pohyby robotu ve všech pohybových úlohách. Všechny ukazatele programů musí být posunuty před restartem vykonávání programu.

Příklad 3

```
SystemStopAction \Halt;
```

Výsledkem bude vypnutí motorů, zastavení vykonávání programu a pohyby robotu ve všech pohybových úlohách. Před restartem vykonávání programu musí být zapnuty motory (Motors on).

Argumenty

```
SystemStopAction [\Stop] [\StopBlock] [\Halt]
```

`[\Stop]`

Datový typ: `switch`

`\Stop` se používá pro zastavení vykonávání programu a pohyby robotu ve všech pohybových úlohách. Žádná konkrétní činnost není nutná před restartováním vykonávání programu.

`[\StopBlock]`

Datový typ: `switch`

`\StopBlock`se používá pro zastavení vykonávání programu a pohyby robotu ve všech pohybových úlohách. Všechny ukazatele programů musí být posunuty před restartem vykonávání programu.

`[\Halt]`

Datový typ: `switch`

Výsledkem `\Halt` bude vypnutí motorů, zastavení vykonávání programu a pohyby robotu ve všech pohybových úlohách. Před restartem vykonávání programu musí být zapnuty motory (Motors on).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Jestliže robot provádí kruhový pohyb během `SystemStopAction \StopBlock`, ukazatel programu a robot musí být posunuty na začátek kruhového pohybu před restartem vykonávání programu.

Vykonávání programu

`SystemStopAction` se používá pro zastavení systému robotu různými způsoby podle závažnosti chyby nebo problému. Vykonávání programu je zastaveno v provádějící úloze, jestliže úloha je normální úlohou.

Jestliže vykonávání `SystemStopAction` probíhá ve statické nebo polostatické úloze, vykonávání programu se zastaví u všech normálních úloh, ale pokračuje pro tuto úlohu. Více podrobností o deklaraci úloh najdete v dokumentaci k systémovým parametrům.

Syntaxe

```
SystemStopAction
  [ '\'Stop ]
  | [ '\'StopBlock ]
  | [ '\'Halt ]';'
```

Související informace

Pro informace o	Viz
Ukončit provádění programu	*Stop - Ukončuje vykonávání programu na str 731*
Ukončit vykonávání programu	*EXIT - Ukončuje vykonávání programu na str 213*
Pouze zastavit pohyby robotu	*StopMove - Zastavuje pohyby robotu na str 736*
Zapsat chybovou zprávu	*ErrLog - Zapsat chybovou zprávu na str 203*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.279 TEST - Závidí na hodnotě výrazu ...

Použití

TEST se používá, když mají být vykonány různé instrukce v závislosti na hodnotě výrazu nebo dat.

Jestliže tam není příliš mnoho alternativ, je možné použít také instrukci IF..ELSE.

Základní příklady

Následující příklad názorně ukazuje instrukci TEST:

Příklad 1

```
TEST reg1
CASE 1,2,3 :
  routine1;
CASE 4 :
  routine2;
DEFAULT :
  TPWrite "Illegal choice";
  Stop;
ENDTEST
```

Různé instrukce jsou vykonávány podle hodnoty reg1. Jestliže hodnota je 1, 2 nebo 3, vykoná se routine1. Jestliže hodnota je 4, vykoná se routine2. Jinak je chybová zpráva vytištěna a vykonávání se zastaví.

Argumenty

```
TEST Test data {CASE Test value {, Test value} : ...} [ DEFAULT:
     ...] ENDTEST
```

`Test data`

Datový typ: Všechny

Data nebo výraz, se kterými bude porovnávána testovací hodnota.

`Test value`

Datový typ: Stejný jako `test data`

Hodnota, kterou musí mít testovací data pro připojené instrukce, aby mohly být vykonány.

Vykonávání programu

Testovací data jsou porovnávána s testovacím hodnotami v první podmínce CASE. Jestliže porovnání je true, potom jsou připojené instrukce vykonány. Potom pokračuje vykonávání programu s instrukcí následující ENDTEST.

Jestliže není splněna první podmínka CASE, potom jsou testovány jiné podmínky CASE a tak dále. Jestliže žádná z podmínek není splněna, potom jsou vykonány instrukce spojené s DEFAULT (jestliže je přítomna).

Syntaxe

TEST <expression>

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
{ CASE <test value> { ',' <test value> } ':'
  <statement list> }
[ DEFAULT ':'
  <statement list> ]
ENDTEST
```

Související informace

Pro informace o	Viz
Výrazy	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.280 TestSignDefine - Definovat testovací signál

Použití

`TestSignDefine` se používá k definování jednoho testovacího signálu pro systém pohybu robotu.

Testovací signál neustále zrcadlí některý určený stream pohybových dat. Například, reference momentu pro některé určené osy. Aktuální hodnota v určitém čase může být přečtena v RAPIDu s funkcí `TestSignRead`.

Může být dosaženo pouze testovacích signálů pro externí osy. Testovací signály jsou dostupné také na vyžádání pro osy robotu a pro předem nedefinované testovací signály pro externí osy.

Základní příklady

Následující příklad názorně ukazuje instrukci `TestSignDefine`:

Příklad 1

```
TestSignDefine 1, resolver_angle, Orbit, 2, 0.1;
```

Testovací signál `resolver_angle` připojený ke kanálu `1` bude dávat hodnotu úhlu rozkladače pro externí osu `2` na manipulátoru `orbit,` vzorkovanou při 100 ms.

Argumenty

```
TestSignDefine Channel SignalId MechUnit Axis SampleTime
```

`Channel`

Datový typ: `num`

Čísla kanálů 1-12, která budou použita pro test signál. Stejné číslo musí být použito ve funkci `TestSignRead` pro čtení aktuální hodnoty test signálu.

`SignalId`

Datový typ: `testsignal`

Jméno nebo číslo test signálu. Viz předdefinované konstanty popsané v datovém typu `testsignal`.

`MechUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`Axis`

Datový typ: `num`

Číslo osy v rámci mechanické jednotky.

`SampleTime`

Datový typ: `num`

Vzorkovací čas v sekundách.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

U vzorkovacího času < 0,004 s funkce `TestSignRead` vrací střední hodnotu naposledy dostupných interních vzorků, jak je vidět v tabulce dole.

Vzorkovací čas v sekundách	Výsledek od TestSignRead
0	Střední hodnota posledních 8 vzorků generovaných každou 0,5 ms
0,001	Střední hodnota posledních 4 vzorků generovaných každé 1 ms
0.002	Střední hodnota posledních 2 vzorků generovaných každé 2 ms
Větší nebo roven 0,004.	Krátkodobá hodnota generovaná v určeném vzorkovacím čase
0,1	Krátkodobá hodnota generovaná v určeném vzorkovacím čase 100 ms

Vykonávání programu

Definice test signálu je aktivována a systém robotu začíná vzorkovat test signál.

Vzorkování test signálu je aktivní až do:

- Vykonává se nová instrukce `TestSignDefine` pro aktuální kanál.
- Všechny test signály jsou deaktivovány s vykonáním instrukce `TestSignReset`.
- Všechny test signály jsou deaktivovány při **Restartu** systému.

Řešení chyb

Jestliže je chyba v parametru `MechUnit`, proměnná `ERRNO` se nastaví na `ERR_UNIT_PAR`. Jestliže je chyba v parametru `Axis`, potom je `ERRNO` nastavena na `ERR_AXIS_PAR`.

Syntaxe

```
TestSignDefine
  [ Channel ':=' ] < expression (IN) of num>' ,'
  [ SignalId' :=' ] < expression (IN) of testsignal> ','
  [ MechUnit' :=' ] < variable (VAR) of mecunit> ','
  [ Axis ':=' ] < expression (IN) of num> ','
  [ SampleTime' :=' ] < expression (IN) of num > ';'
```

Související informace

Pro informace o	Viz
Testovací signál	*testsignal - Testovací signál na str 1609*
Přečíst testovací signál	*TestSignRead - Přečíst hodnotu testovacího signálu na str 1366*
Resetovat testovací signály	*TestSignReset - Resetovat všechny definice testovacích signálů na str 778*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.281 TestSignReset - Resetovat všechny definice testovacích signálů

Použití

`TestSignReset` se používá k deaktivaci všech dříve definovaných test signálů.

Základní příklady

Následující příklad názorně ukazuje instrukci `TestSignReset`:

Příklad 1

```
TestSignReset;
```

Deaktivovat všechny dříve definované testovací signály.

Vykonávání programu

Definice všech test signálů jsou deaktivovány a systém robotu zastavuje vzorkování všech test signálů.

Vzorkování definovaných test signálů je aktivní až do:

- **Restart** systému
- Vykonání této instrukce `TestSignReset`

Syntaxe

```
TestSignReset';'
```

Související informace

Pro informace o	Viz
Testovací signál	*testsignal - Testovací signál na str 1609*
Definovat testovací signál	*TestSignDefine - Definovat testovací signál na str 776*
Přečíst testovací signál	*TestSignRead - Přečíst hodnotu testovacího signálu na str 1366*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.282 TextTabInstall - Instalace textové tabulky

Použití

Použijte `TextTabInstall`, chcete-li Instalovat textovou tabulku do systému.

Základní příklady

Následující příklad názorně ukazuje instrukci `TextTabInstall`:

Příklad 1

```
! System Module with Event Routine to be executed at event
! POWER ON, RESET or START

PROC install_text()
  IF TextTabFreeToUse("text_table_name") THEN
    TextTabInstall "HOME:/text_file.eng";
  ENDIF
ENDPROC
```

Při prvním vykonávání událostní rutiny `install_text` funkce `TextTabFreeToUse` vrací TRUE a textový soubor `text_file.eng` je instalován do systému. Potom mohou být instalované textové řetězce získány ze systému do RAPIDu funkcemi `TextTabGet` a `TextGet`.

Příště, když je událostní rutina `install_text` vykonávána, funkce `TextTabFreeToUse` vrací FALSE a instalace se **neopakuje**.

Argumenty

```
TextTabInstall File
```

`File`

Datový typ: `string`

Cesta souboru a jméno souboru k souboru, který obsahuje textové řetězce, které mají být instalovány do systému.

Omezení

Omezení pro instalaci textových tabulek (testové zdroje) v systému:

- Není možné instalovat stejnou textovou tabulku do systému více než jedenkrát.
- Není možné odinstalovat (uvolnit) jednoduchou textovou tabulku ze systému. Jediným způsobem, jak odinstalovat textové tabulky ze systému, je restartovat řadič pomocí režimu restartu **Resetovat systém**. Všechny textové tabulky (definované systémem a uživatelem) budou potom odinstalovány.

Řešení chyb

Jestliže soubor v instrukci `TextTabInstall` není možné otevřít, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEOPEN`. Tato chyba může být potom ošetřena v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.282 TextTabInstall - Instalace textové tabulky

RobotWare - OS

Pokračování

Syntaxe

```
TextTabInstall
  [ File ':=' ] < expression (IN) of string >';'
```

Související informace

Pro informace o	Viz
Test, jestli je textová tabulka volná	*TextTabFreeToUse - Otestovat, jestli je textová tabulka volná na str 1370*
Formáty textových souborů	*Technická referenční příručka - RAPID kernel*
Získat číslo textové tabulky	*TextTabGet - Získat číslo textové tabulky na str 1372*
Vzít text ze systémových textových tabulek	*TextGet - Vzít text ze systémových textových tabulek na str 1368*
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.283 TPErase - Vymaže text vytištěný na FlexPendantu

Použití

`TPErase` (*FlexPendant Erase*) is se používá na vyčištění displeje FlexPendantu.

Základní příklady

Následující příklad názorně ukazuje instrukci `TPErase`:

Příklad 1

```
TPErase;
TPWrite "Execution started";
```

Displej FlexPendantu je vyčištěn před napsáním `Execution started`.

Vykonávání programu

Displej FlexPendantu se kompletně vyčistí od veškerého textu. Až bude text zapisován příště, bude vložen na řádku až zcela nahoře na displeji.

Syntaxe

```
TPErase;
```

Související informace

Pro informace o	Viz
Zapisování na FlexPendant	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.284 TPReadDnum - Čte číslo z FlexPendantu

Použití

`TPReadDnum` (*FlexPendant Read Numerical*) se používá pro čtení čísla z FlexPendantu

Základní příklady

Následující příklad názorně ukazuje instrukci `TPReadDnum`:

Příklad 1

```
VAR dnum value;

TPReadDnum value, "How many units should be produced?";
```

Text `How many units should be produced?` je zapsán na displej FlexPendantu. Vykonávání programu čeká na vložení čísla z numerické klávesnice na FlexPendant. Číslo se uloží do `value`.

Argumenty

```
TPReadDnum TPAnswer TPText [\MaxTime][\DIBreak] [\DIPassive]
     [\DOBreak] [\DOPassive] [\BreakFlag]
```

`TPAnswer`

Datový typ: `dnum`

Proměnná, pro kterou je vrácen vstup čísla přes FlexPendant.

`TPText`

Datový typ: `string`

Informační text, který bude napsán na FlexPendantu (max 80 znaků, 40 znaků na řádce)

`[\MaxTime]`

Datový typ: `num`

Max množství času, kdy vykonávání programu čeká. Jestliže během této doby není vloženo žádné číslo, program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_MAXTIME` může být použita pro testování, jestli uplynula max doba nebo nikoliv.

`[\DIBreak]`

Digital Input Break

Datový typ: `signaldi`

Digitální signál, který může přerušit dialog operátora. Jestliže není vloženo žádné číslo, když signál je nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DIBREAK` může být použita pro testování, jestli uplynula max doba nebo nikoliv.

`[\DIPassive]`

Digital Input Passive

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DIBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DIBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DIBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\DOBreak]`

Digital Output Break

Datový typ: `signaldo`

Digitální signál, který podporuje žádost o ukončení od jiných úloh. Jestliže není zvoleno žádné tlačítko, když signál je nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DOBREAK` může být použita pro testování, jestli už toto nastalo nebo nikoliv.

`[\DOPassive]`

Digital Output Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DOBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DOBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DOBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\BreakFlag]`

Datový typ: `errnum`

Proměnná, která drží chybový kód, jestliže se používá `MaxTime`, `DIBreak` nebo `DOBreak`. Jestliže tato volitelná proměnná je vypuštěna, bude vykonán chybový handler. Konstanty `ERR_TP_MAXTIME`, `ERR_TP_DIBREAK` a `ERR_TP_DOBREAK` mohou být použity pro volbu důvodu.

Vykonávání programu

Informační text se píše vždy na novou řádku. Jestliže displej je plný textu, toto tělo textu je nejprve posunuto o jednu řádku nahoru. Nad nově zapsaným textem může být až 7 řádek.

Vykonávání programu čeká na napsání čísla na numerické klávesnici (následováno Enterem nebo OK) nebo instrukce je přerušena vypršením času nebo činností signálu..

Reference na `TPReadFK` o popisu souběžného požadavku `TPReadFK` nebo `TPReadDnum` na FlexPendantu ze stejné nebo jiných programových úloh.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.284 TPReadDnum - Čte číslo z FlexPendantu

RobotWare - OS

Pokračování

Řešení chyb

Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`. Při používání tohoto signálu je systémová proměnná `ERRNO` nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.

Při vypršení času (parametr `\MaxTime`) před vstupem operátora je systémová proměnná `ERRNO` nastavena na `ERR_TP_MAXTIME` a vykonávání pokračuje v chybovém handleru.

Jestliže je nastaven digitální vstup (parametr `\DIBreak`) před vstupem operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DIBREAK` a vykonávání pokračuje v chybovém handleru.

Jestliže se objevil digitální výstup (parametr `\DOBreak`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DOBREAK` a vykonávání pokračuje v chybovém handleru.

Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná `ERRNO` je nastavena na `ERR_TP_NO_CLIENT` a vykonávání pokračuje v chybovém handleru.

Tyto situace mohou být potom ošetřeny chybovým handlerem.

Syntaxe

```
TPReadDnum
  [TPAnswer':='] <var or pers (INOUT) of dnum>','
  [TPText':='] <expression (IN) of string>
  ['\'MaxTime':=' <expression (IN) of num>]
  ['\'DIBreak':=' <variable (VAR) of signaldi>]
  ['\'DIPassive]
  ['\'DOBreak':=' <variable (VAR) of signaldo>]
  ['\'DOPassive]
  ['\'BreakFlag':=' <var or pers (INOUT) of errnum>] ';'
```

Související informace

Pro informace o	Viz
Zápis a čtení na FlexPendantu	*Technická referenční příručka - Přehled RAPID*
Vkládání čísla na FlexPendantu	*Návod k použití - IRC5 s jednotkou FlexPendant*
Příklady, jak používat argumenty MaxTime, DIBreak a BreakFlag	*TPReadFK - Čte funkční klávesy na str 785*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.285 TPReadFK - Čte funkční klávesy

Použití

`TPReadFK` (*FlexPendant Read Function Key*) se používá pro psaní textu na funkčních klávesách a pro zjišťování, která klávesa je stisknuta.

Základní příklady

Následující příklad názorně ukazuje instrukci `TPReadFK`:

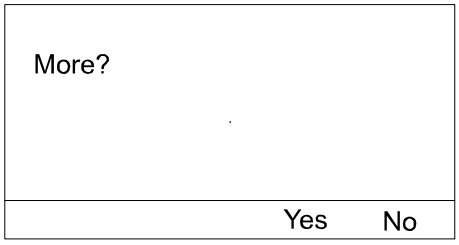
Viz také *Další příklady na str 787*.

Příklad 1

```
TPReadFK reg1, "More?", stEmpty, stEmpty, stEmpty, "Yes", "No";
```

Text `More?` je napsán na displej FlexPendantu a funkční klávesy 4 a 5 jsou aktivovány pomocí textových řetězců `Yes` a `No` (viz obrázek dole). Vykonávání programu čeká na stisknutí jedné z funkčních kláves 4 a 5. Jinými slovy, pro `reg1` bude přiděleno 4 nebo 5 podle toho, která klávesa je stisknuta.

Obrázek ukazuje, že operátor může vložit informace přes funkční klávesy.



xx0500002345

Argumenty

```
TPReadFK TPAnswer TPText TPFK1 TPFK2 TPFK3 TPFK4 TPFK5 [\MaxTime]
    [\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive] [\BreakFlag]
```

`TPAnswer`

Datový typ: `num`

Proměnná, pro kterou je vrácena numerická hodnota 1..5, podle toho, která klávesa je stisknuta. Jestliže je stisknuta funkční klávesa 1, potom je vrácena 1, a tak dále.

`TPText`

Datový typ: `string`

Informační text, který bude napsán na displej (max 80 znaků, 40 znaků na řádce)

`TPFKx`

Function key text

Datový typ: `string`

Text, který bude zapsán na příslušnou funkční klávesu (max 45 znaků). `TPFK1` je klávesa zcela vlevo.

Funkční klávesy bez textu jsou určeny předdefinovanou řetězcovou konstantou `stEmpty` s prázdnou hodnotou řetězce (“”).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.285 TPReadFK - Čte funkční klávesy

RobotWare - OS

Pokračování

[\MaxTime]

Datový typ: num

Max množství času v sekundách, kdy vykonávání programu čeká. Jestliže během této doby není stisknuta žádná funkční klávesa, program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_MAXTIME může být použita pro testování, jestli uplynula max doba nebo nikoliv.

[\DIBreak]

Digital Input Break

Datový typ: signaldi

Digitální signál, který může přerušit dialog operátora. Jestliže není stisknuta žádná funkční klávesa, když signál je nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_DIBREAK může být použita pro testování, jestli toto už nastalo nebo nikoliv.

[\DIPassive]

Digital Input Passive

Datový typ: switch

Tento přepínač potlačuje standardní chování při použití volitelného argumentu DIBreak. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný BreakFlag), když je signál DIBreak nastaven na 0 (nebo již je 0). Konstantu ERR_TP_DIBREAK je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

[\DOBreak]

Digital Output Break

Datový typ: signaldo

Digitální signál, který podporuje žádost o ukončení od jiných úloh. Jestliže není zvoleno žádné tlačítko, když signál je nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_DOBREAK může být použita pro testování, jestli už toto nastalo nebo nikoliv.

[\DOPassive]

Digital Output Passive

Datový typ: switch

Tento přepínač potlačuje standardní chování při použití volitelného argumentu DOBreak. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný BreakFlag), když je signál DOBreak nastaven na 0 (nebo již je 0). Konstantu ERR_TP_DOBREAK je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

[\BreakFlag]

Datový typ: errnum

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Proměnná, která bude držet chybový kód, jestliže se použije `MaxTime`, `DIBreak` nebo `DOBreak`. Jestliže tato volitelná proměnná je vypuštěna, bude vykonán chybový handler. Konstanty `ERR_TP_DIBREAK`, `ERR_TP_MAXTIME`, a `ERR_TP_DOBREAK` mohou být použity pro volbu důvodu.

Vykonávání programu

Informační text se píše vždy na novou řádku. Jestliže displej je plný textu, toto tělo textu je nejprve posunuto o jednu řádku nahoru. Nad nově zapsaným textem může být až 7 řádek.

Text je zapsán na příslušné funkční klávesy.

Vykonávání programu čeká na stisknutí jedné z aktivovaných funkčních kláves.

Popis souběžného požadavku `TPReadFK` nebo `TPReadNum` na FlexPendantu (požadavek TP) ze stejné nebo jiných programových úloh:

- Nový TP požadavek od jiných programových úloh nebude ústředním bodem (nové vložení do fronty).
- Nový TP požadavek od TRAP ve stejné programové úloze bude středem pozornosti (staré vložení do fronty)
- Zastavení programu je středem pozornosti (staré vložení do fronty)
- Nový TP požadavek ve stavu zastavení je středem pozornosti (staré vložení do fronty)

Další příklady

Více příkladů jak používat instrukci `TPReadFK` je názorně uvedeno dole.

Příklad 1

```
VAR errnum errvar;
...
TPReadFK reg1, "Go to service position?", stEmpty, stEmpty, stEmpty,
     "Yes","No"
\MaxTime:= 600
  \DIBreak:= di5\BreakFlag:= errvar;
IF reg1 = 4 OR errvar = ERR_TP_DIBREAK THEN
  MoveL service, v500, fine, tool1;
  Stop;
ENDIF
IF errvar = ERR_TP_MAXTIME EXIT;
```

Robot je posunut do servisní pozice, jestliže funkční klávesa (`"Yes"`) je stisknuta nebo jestliže vstup 5 je aktivován. Jestliže není dána žádná odpověď během 10 sekund, potom je vykonávání ukončeno.

Řešení chyb

Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`. Při používání tohoto signálu je systémová proměnná `ERRNO` nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.285 TPReadFK - Čte funkční klávesy

RobotWare - OS

Pokračování

Jestliže se objevilo vypršení času (parametr \MaxTime) před vstupem od operátora, systémová proměnná ERRNO je nastavena na ERR_TP_MAXTIME a vykonávání pokračuje v chybovém handleru.

Jestliže je nastaven digitální vstup (parametr \DIBreak) před vstupem operátora, systémová proměnná ERRNO je nastavena na ERR_TP_DIBREAK a vykonávání pokračuje v chybovém handleru.

Jestliže se objevil digitální výstup (parametr \DOBreak) před vstupem od operátora, systémová proměnná ERRNO je nastavena na ERR_TP_DOBREAK a vykonávání pokračuje v chybovém handleru.

Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná ERRNO je nastavena na ERR_TP_NO_CLIENT a vykonávání pokračuje v chybovém handleru.

Tyto situace mohou být potom ošetřeny chybovým handlerem.

Omezení

Vyhněte se používání přílišných hodnot pro parametr vypršení času \MaxTime , když je často vykonáván TPReadFK, např. ve smyčce. Výsledkem může být nepředvídané chování činnosti systému, jak je zpomalení odezvy FlexPendantu.

Předdefinovaná data

```
CONST string stEmpty := "";
```

Předdefinovaná konstanta stEmpty se může používat pro funkční klávesy bez textu.

Syntaxe

```
TPReadFK
  [TPAnswer ':='] <var or pers (INOUT) of num>','
  [TPText ':='] <expression (IN) of string>','
  [TPFK1 ':='] <expression (IN) of string>','
  [TPFK2 ':='] <expression (IN) of string>','
  [TPFK3 ':='] <expression (IN) of string>','
  [TPFK4 ':='] <expression (IN) of string>','
  [TPFK5 ':='] <expression (IN) of string>
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>]';'
```

Související informace

Pro informace o	Viz
Zápis a čtení na FlexPendantu	*Technická referenční příručka - Přehled RAPID*
Odpovídání přes FlexPendant	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.286 TPReadNum - Čte číslo z FlexPendantu

Použití

`TPReadNum` (*FlexPendant Read Numerical*) se používá pro čtení čísla z FlexPendantu.

Základní příklady

Následující příklad názorně ukazuje instrukci `TPReadNum`:

Viz také *Další příklady na str 790*.

Příklad 1

```
TPReadNum reg1, "How many units should be produced?";
```

Text `How many units should be produced?` je zapsán na displej FlexPendantu. Vykonávání programu čeká na vložení čísla z numerické klávesnice na FlexPendant. Číslo se uloží do `reg1`.

Argumenty

```
TPReadNum TPAnswer TPText [\MaxTime][\DIBreak] [\DIPassive]
    [\DOBreak] [\DOPassive] [\BreakFlag]
```

`TPAnswer`

Datový typ: `num`

Proměnná, pro kterou je vrácen vstup čísla přes FlexPendant.

`TPText`

Datový typ: `string`

Informační text, který bude napsán na FlexPendantu (max 80 znaků, 40 znaků na řádce).

`[\MaxTime]`

Datový typ: `num`

Max množství času, kdy vykonávání programu čeká. Jestliže během této doby není vloženo žádné číslo, program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_MAXTIME` může být použita pro testování, jestli uplynula max doba nebo nikoliv.

`[\DIBreak]`

Digital Input Break

Datový typ: `signaldi`

Digitální signál, který může přerušit dialog operátora. Jestliže není vloženo žádné číslo, když signál je nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DIBREAK` může být použita pro testování, jestli uplynula max doba nebo nikoliv.

`[\DIPassive]`

Digital Input Passive

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.286 TPReadNum - Čte číslo z FlexPendantu
RobotWare - OS
Pokračování

Tento přepínač potlačuje standardní chování při použití volitelného argumentu DIBreak. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný BreakFlag), když je signál DIBreak nastaven na 0 (nebo již je 0). Konstantu ERR_TP_DIBREAK je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

[\DOBreak]

Digital Output Break

Datový typ: signaldo

Digitální signál, který podporuje žádost o ukončení od jiných úloh. Jestliže není zvoleno žádné tlačítko, když signál je nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_DOBREAK může být použita pro testování, jestli už toto nastalo nebo nikoliv.

[\DOPassive]

Digital Output Passive

Datový typ: switch

Tento přepínač potlačuje standardní chování při použití volitelného argumentu DOBreak. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný BreakFlag), když je signál DOBreak nastaven na 0 (nebo již je 0). Konstantu ERR_TP_DOBREAK je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

[\BreakFlag]

Datový typ: errnum

Proměnná, která drží chybový kód, jestliže se používá MaxTime, DIBreaknebo DOBreak. Jestliže tato volitelná proměnná je vypuštěna, bude vykonán chybový handler. Konstanty ERR_TP_MAXTIME, ERR_TP_DIBREAK a ERR_TP_DOBREAK mohou být použity pro volbu důvodu.

Vykonávání programu

Informační text se píše vždy na novou řádku. Jestliže displej je plný textu, toto tělo textu je nejprve posunuto o jednu řádku nahoru. Nad nově zapsaným textem může být až 7 řádek.

Vykonávání programu čeká na napsání čísla na numerické klávesnici (následováno Enterem nebo OK) nebo instrukce je přerušena vypršením času nebo činností signálu.

Reference na TPReadFK o popisu souběžného požadavku TPReadFK nebo TPReadNum na FlexPendantu ze stejné nebo jiných programových úloh.

Další příklady

Více příkladů jak používat instrukci TPReadNum je názorně uvedeno dole.

Příklad 1

```
TPReadNum reg1, "How many units should be produced?";
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
FOR i FROM 1 TO reg1 DO
  produce_part;
ENDFOR
```

Text `How many units should be produced?` je napsán na displej FlexPendantu. Rutina `produce_part` je potom opakována tolikrát, kolik je vloženo přes FlexPendant.

Řešení chyb

Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`. Při používání tohoto signálu je systémová proměnná `ERRNO` nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.

Při vypršení času (parametr `\MaxTime`) před vstupem operátora je systémová proměnná `ERRNO` nastavena na `ERR_TP_MAXTIME` a vykonávání pokračuje v chybovém handleru.

Jestliže je nastaven digitální vstup (parametr `\DIBreak`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DIBREAK` a vykonávání pokračuje v chybovém handleru.

Jestliže je nastaven digitální výstup (parametr `\DOBreak`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DOBREAK` a vykonávání pokračuje v chybovém handleru.

Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná `ERRNO` je nastavena na `ERR_TP_NO_CLIENT` a vykonávání pokračuje v chybovém handleru.

Tyto situace mohou být potom ošetřeny chybovým handlerem.

Syntaxe

```
TPReadNum
  [TPAnswer':='] <var or pers (INOUT) of num>','
  [TPText':='] <expression (IN) of string>
  ['\'MaxTime':=' <expression (IN) of num>]
  ['\'DIBreak':=' <variable (VAR) of signaldi>]
  ['\'DIPassive]
  ['\'DOBreak':=' <variable (VAR) of signaldo>]
  ['\'DOPassive]
  ['\'BreakFlag':=' <var or pers (INOUT) of errnum>] ';'
```

Související informace

Pro informace o	Viz
Zápis a čtení na FlexPendantu	*Technická referenční příručka - Přehled RAPID*
Vkládání čísla na FlexPendantu	*Návod k použití - IRC5 s jednotkou FlexPendant*
Příklady, jak používat argumenty MaxTime, DIBreak a BreakFlag	*TPReadFK - Čte funkční klávesy na str 785*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.287 TPShow - Okno přepínače na FlexPendantu

Použití

`TPShow` (*FlexPendant Show*) se používá pro výběr okna FlexPendantu z RAPIDu.

Základní příklady

Následující příklad názorně ukazuje instrukci `TPShow`:

Příklad 1

```
TPShow TP_LATEST;
```

Naposledy použité okno FlexPendantu před aktuálním oknem FlexPendantu bude aktivní po vykonání této instrukce.

Argumenty

```
TPShow Window
```

`Window`

Datový typ: `tpnum`

Okno TP_LATEST bude ukazovat naposledy použité okno FlexPendantu před aktuálním oknem FlexPendantu.

Předdefinovaná data

```
CONST tpnum TP_LATEST := 2;
```

Vykonávání programu

Zvolené okno FlexPendantu bude aktivováno.

Syntaxe

```
TPShow
  [Window':='] <expression (IN) of tpnum> ';'
```

Související informace

Pro informace o	Viz
Komunikace pomocí FlexPendantu	*Technická referenční příručka - Přehled RAPID*
Číslo okna FlexPendantu	*tpnum - Číslo okna FlexPendantu na str 1616*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.288 TPWrite - Zapisuje na FlexPendant

Použití

`TPWrite` (*FlexPendant Write*) se používá pro zapisování textu na FlexPendant. Může být zapsána hodnota konkrétních dat, stejně tak jako text.

Základní příklady

Následující příklady názorně ukazují instrukci `TPWrite`:

Příklad 1

```
TPWrite "Execution started";
```

Text `Execution started` je zapsán na FlexPendant.

Příklad 2

```
TPWrite "No of produced parts="\Num:=reg1;
```

Jestliže například `reg1` drží hodnotu 5, potom je na FlexPendant zapsán text `No of produced parts=5`.

Příklad 3

```
VAR string my_robot;
...
my_robot := RobName();
IF my_robot="" THEN
  TPWrite "This task does not control any TCP robot";
ELSE
  TPWrite "This task controls TCP robot with name "+ my_robot;
ENDIF
```

Zapsat na FlexPendant jméno TCP robotu, který je řízen z této programové úlohy. Jestliže není řízen žádný TCP robot, zapište, že tato úloha neřídí žádný robot.

Argumenty

```
TPWrite String [\Num] | [\Bool] | [\Pos] | [\Orient] | [\Dnum]
```

`String`

Datový typ: `string`

Textový řetězec, který bude zapsán (max 80 znaků, 40 znaků na řádce).

`[\Num]`

Numeric

Datový typ: `num`

Data, jejichž numerická hodnota bude zapsána po textovém řetězci.

`[\Bool]`

Boolean

Datový typ: `bool`

Data, jejichž logická hodnota bude zapsána po textovém řetězci.

`[\Pos]`

Position

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.288 TPWrite - Zapisuje na FlexPendant
RobotWare - OS
Pokračování

Datový typ: pos

Data, jejichž pozice bude zapsána po textovém řetězci.

[\Orient]

Orientation

Datový typ: orient

Data, jejichž orientace bude zapsána po textovém řetězci.

[\Dnum]

Numeric

Datový typ: dnum

Data, jejichž numerická hodnota bude zapsána po textovém řetězci.

Vykonávání programu

Text zapsaný na FlexPendant vždy začíná na nové řádce. Když je displej plný textu (11 řádek), potom je tento text nejprve posunut nahoru o jednu řádku.

Když je použit jeden z argumentů \Num, \Dnum, \Bool, \Pos, nebo \Orient , potom je tato hodnota nejprve převedena do textového řetězce, předtím než je přidána do prvního řetězce. Převod z hodnoty na textový řetězec probíhá následovně:

Argument	Hodnota	Textový řetězec
\Num	23	"23"
\Num	1,141367	"1.14137"
\Bool	TRUE	"TRUE"
\Pos	[1817.3,905.17,879.11]	"[1817.3,905.17,879.11]"
\Orient	[0.96593,0,0.25882,0]	"[0.96593,0,0.25882,0]"
\Dnum	4294967295	"4294967295"

Hodnota je převedena na řetězec se standardním formátem RAPID. To znamená, v principu, 6 významných číslic. Jestliže desetinná část je méně než 0,000005 nebo více než 0,999995, číslo je zaokrouhleno na celé číslo.

Omezení

Argumenty \Num, \Dnum, \Bool, \Pos, a \Orient jsou neslučitelné a tady nemohou být použity současně ve stejné instrukci.

Syntaxe

```
TPWrite
  [TPText':='] <expression (IN) of string>
  ['\'Num':=' <expression (IN) of num> ]
  | ['\'Bool':=' <expression (IN) of bool> ]
  | ['\'Pos':=' <expression (IN) of pos> ]
  | ['\'Orient':=' <expression (IN) of orient> ]
  | ['\'Dnum':=' <expression (IN) of dnum> ]';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Vyčištění a čtení FlexPendantu	*Technická referenční příručka - Přehled RAPID*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.289 TriggC - Kruhový pohyb robotu s událostmi

Použití

`TriggC` (*Trigg Circular*) se používá pro nastavení výstupních signálů a/nebo provádění rutin přerušení na pevných pozicích ve stejném čase, kdy se robot pohybuje na kruhové dráze.

Jedna nebo více (max. 8) událostí může být definováno pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed`, nebo `TriggRampAO`, a poté je na tyto definice odkázáno v instrukci `TriggC`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggC`:

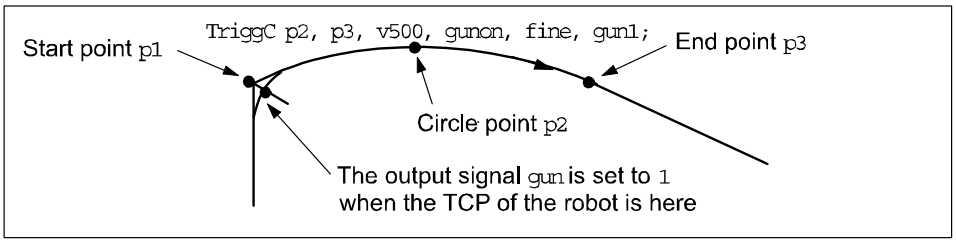
Viz také *Další příklady na str 800*.

Příklad 1

```
VAR triggdata gunon;

TriggIO gunon, 0 \Start \DOp:=gun, 1;
MoveL p1, v500, z50, gun1;
TriggC p2, p3, v500, gunon, fine, gun1;
```

Digitální výstupní signál `gun` je nastaven, když TCP robotu přechází přes midpoint rohové dráhy bodu `p1`.

Obrázek ukazuje příklad I/O události pevné pozice.



xx0500002267

Argumenty

```
TriggC [\Conc] CirPoint ToPoint [\ID] Speed [\T] Trigg_1 |
     TriggArray{*} [\T2] [\T3] [\T4] [\T5] [\T6] [\T7] [\T8] Zone
     [\Inpos] Tool [\WObj] [ \Corr ] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datový typ:`switch`

Následné instrukce jsou vykonány během pohybu robotu. Argument se obvykle nepoužívá, ale může se použít kvůli zabránění nechtěným zastavením způsobeným přetíženým CPU při používání průjezdných bodů. To je vhodné, když naprogramované body jsou velmi blízko sebe při velkých rychlostech. Argument

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

je také užitečný, když například komunikace s externím vybavením a synchronizace mezi externím vybavením a pohybem robotu nejsou žádoucí. Může se také použít k ladění vykonávání dráhy robotu, aby se předešlo varování 50024 Závada rohové dráhy nebo chybě 50082 Limit zpomalení.

Použitím argumentu `\Conc` je počet pohybových instrukcí za sebou omezen na 5. V programové sekci, která obsahuje `StorePath-RestoPath`, nejsou pohybové instrukce s argumentem `\Conc` povoleny.

Jestliže tento argument je vypuštěn a `ToPoint` není stop bodem, následná instrukce je vykonána předtím, než robot dosáhne naprogramované zóny.

Tento argument se nemůže používat v koordinovaném synchronizovaném pohybu v systému MultiMove.

`CirPoint`

Datový typ: `robtarget`

Kruhový bod robotu. Viz instrukce `MoveC`, kde je podrobnější popis kruhového pohybu. Kruhový bod je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`ToPoint`

Datový typ: `robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Trigg_1`

Datový typ: `triggdata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.289 TriggC - Kruhový pohyb robotu s událostmi
RobotWare - OS
Pokračování

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`TriggArray`

Trigg Data Array Parameter

Datový typ: `triggdata`

Proměnná pole, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

Omezení je 25 elementů v poli a musí být definováno 1 až 25 spouštěcích podmínek.

Není možné používat volitelné parametry `T2`, `T3`, `T4`, `T5`, `T6`, `T7` nebo `T8` ve stejném čase, kdy je používán argument `TriggArray`.

`[ \T2 ]`

Trigg 2

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T3 ]`

Trigg 3

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T4 ]`

Trigg 4

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T5 ]`

Trigg 5

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheck`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T6 ]`

Trigg 6

Datový typ: `triggdata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T7 ]`

Trigg 7

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T8 ]`

Trigg 8

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`Zone`

Datový typ: `zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`[ \Inpos ]`

In position

Datový typ: `stoppointdata`

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určené pozice (destinace).

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být stanoven pro lineární pohyb ve vztahu k pracovnímu objektu, který bude proveden.

`[ \Corr ]`

Correction

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.289 TriggC - Kruhový pohyb robotu s událostmi
RobotWare - OS
Pokračování

Korekční data zapsaná do vstupu korekcí instrukcí `CorrWrite` budou přidána k pozici dráhy a destinace, jestliže tento argument je přítomen.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

V instrukci `MoveC` najdete více informací o kruhovém pohybu.

Jelikož spouštěcí podmínky jsou splněny, když je robot umístěn blíž a blíž ke koncovému bodu, definované spouštěcí aktivity jsou provedeny. Spouštěcí podmínky jsou splněny buď v určité vzdálenosti před koncovým bodem instrukce nebo v určité vzdálenosti po bodu startu instrukce nebo v určitém časovém bodu (omezeno na krátký čas) před koncovým bodem instrukce.

Během krokového provádění dopředu jsou I/O aktivity provedeny, ale rutiny přerušení neběží. Během krokového provádění dozadu nejsou prováděny žádné spouštěcí aktivity.

Další příklady

Více příkladů jak používat instrukci `TriggC` je názorně uvedeno dole.

Příklad 1

```
VAR intnum intno1;
VAR triggdata trigg1;
...
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
PROC main()
  ...
  CONNECT intno1 WITH trap1;
  TriggInt trigg1, 0.1 \Time, intno1;
  ...
  TriggC p1, p2, v500, trigg1, fine, gun1;
  TriggC p3, p4, v500, trigg1, fine, gun1;
  ...
  IDelete intno1;
```

Rutina přerušení `trap1` je provedena, když pracovní bod je v pozici 0,1 s před bodem `p2` nebo `p4`.

Příklad 2

```
VAR num Distance:=0;
VAR triggdata trigg_array{25};
VAR signaldo myaliassignaldo;
VAR string signalname;
...
PROC main()
  ...
  FOR i FROM 1 TO 25 DO
    signalname:="do";
    signalname:=signalname+ValToStr(i);
    AliasIO signalname, myaliassignaldo;
    TriggEquip trigg_array{i}, Distance \Start, 0
         \DOp:=myaliassignaldo, SetValue:=1;
    Distance:=Distance+10;
  ENDFOR
  TriggC p1, p2, v500, trigg_array, z30, tool2;
  MoveC p3, p4, v500, z30, tool2;
  ...
```

Digitální výstupní signály `do1` až `do25` jsou během pohybu nastaveny na `p2`. Vzdálenosti mezi nastaveními signálů je 10 mm.

Řešení chyb

Jestliže naprogramovaný argument `ScaleValue` pro stanovený analogový výstupní signál `AOp` v některé z připojených instrukcí `TriggSpeed` je mimo limit pro analogový signál společně s naprogramovaným `Speed` v této instrukci, potom je systémová proměnná `ERRNO` nastavena na `ERR_AO_LIM`.

Jestliže naprogramovaný argument `DipLag` v některé z připojených instrukcí `TriggSpeed` je příliš velký ve vztahu k použitému Event Present Time v systémových parametrech, potom je systémová proměnná `ERRNO` nastavena na `ERR_DIPLAG_LIM`.

Systémová proměnná `ERRNO` může být nastavena na `ERR_NORUNUNIT`, jestliže není kontakt s I/O jednotkou při vstupu do instrukce a použitá triggdata závisí na běžící I/O jednotce, tj. signál se používá v triggdata.

Jestliže počet pohybových instrukcí za sebou pomocí argumentu `\Conc` byl překročen, potom je systémová proměnná `ERRNO` nastavena na `ERR_CONC_MAX`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.289 TriggC - Kruhový pohyb robotu s událostmi
RobotWare - OS
Pokračování

Tyto chyby mohou být zpracovány v chybovém handleru.

Omezení

Všeobecná omezení podle instrukce `MoveC`

Jestliže se aktuální bod startu odchyluje od obvyklého bodu, takže celková délka polohování instrukce `TriggC` je kratší než obvykle, může se stát, že několik nebo všechny spouštěcí podmínky jsou okamžitě splněny na stejné pozici. V takových případech bude sekvence, ve které jsou spouštěcí aktivity prováděny, nedefinována. Programová logika v uživatelském programu nesmí být založena na normální sekvenci spouštěcích aktivit pro „nekompletní pohyb“.

VAROVÁNÍ

Instrukce `TriggC` by nikdy neměla být spouštěna od začátku s robotem v pozici za kruhovým bodem. Jinak robot nevezme naprogramovanou dráhu (polohování kolem kruhové dráhy v jiném směru v porovnání s tím, co bylo naprogramováno).

Syntaxe

```
TriggC
  [ ’\’ Conc ’,’]
  [ CirPoint’ :=’ ] < expression (IN) of robtarget > ’,’
  [ ToPoint’ :=’ ] < expression (IN) of robtarget > ’,’
  [ ’\’ ID ’:=’ < expression (IN) of identno >] ’,’
  [ Speed ’:=’ ] < expression (IN) of speeddata >
  [ ’\’ T ’:=’ < expression (IN) of num > ] ’,’
  [Trigg_1 ’:=’ ] < variable (VAR) of triggdata > |
  [TriggArray ’:=’ ] < array variable {*} (VAR) of triggdata >
  [ ’\’ T2 ’:=’ < variable (VAR) of triggdata > ]
  [ ’\’ T3 ’:=’ < variable (VAR) of triggdata > ]
  [ ’\’ T4 ’:=’ < variable (VAR) of triggdata > ]
  [ ’\’ T5 ’:=’ < variable (VAR) of triggdata > ]
  [ ’\’ T6 ’:=’ < variable (VAR) of triggdata > ]
  [ ’\’ T7 ’:=’ < variable (VAR) of triggdata > ]
  [ ’\’ T8 ’:=’ < variable (VAR) of triggdata > ] ´,’
  [Zone ’:=’ ] < expression (IN) of zonedata >
  [ ’\’ Inpos’ :=’ < expression (IN) of stoppointdata > ]´,’
  [ Tool ’:=’ ] < persistent (PERS) of tooldata >
  [ ’\’ WObj’ :=’ < persistent (PERS) of wobjdata > ]
  [ ’\’ Corr ]
  [ ’\’ TLoad’ :=’ < persistent (PERS) of loaddata > ] ’;’
```

Související informace

Pro informace o	Viz
Lineární pohyb se spouštěči	*TriggL - Lineární pohyby robotu s událostmi na str 839*
Pohyb spoje se spouštěči	*TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Posunout robot kruhově	*MoveC - Pohybuje robotem kruhově na str 358*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Definice spouštěčů	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814* *TriggInt - Definuje přerušení se vztahem k pozici na str 820* *TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804* *TriggRampAO - Definovat událost rampy AO pevné pozice na dráze na str 861* *TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času na str 868*
Manipulace s `triggdata`	*triggdata - Polohovací události, trigg na str 1618* *TriggDataReset - Resetovat obsah v proměnné trigg-data na str 812* *TriggDataCopy - Kopírovat obsah do proměnné trig-gdata na str 810* *TriggDataValid - Zkontrolujte, jestli obsah proměnné triggdata je platný na str 1374*
Zapisuje do vstupu korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
Kruhový pohyb	*Technická referenční příručka - Přehled RAPID*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice dat stop bodu	*stoppointdata - Data stop bodu na str 1590*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoad-Mode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.290 TriggCheckIO - Definuje kontrolu IO v pevné pozici

Použití

`TriggCheckIO` se používá k definování podmínek pro testování hodnoty digitálního -, skupiny digitálních - nebo analogového vstupního a výstupního signálu v pevné pozici podél dráhy pohybu robotu. Jestliže podmínka je splněna, potom nedojde k žádné konkrétní činnosti. Ale pokud není, bude provedena rutina přerušení poté, co robot volitelně zastavil na dráze tak rychle, jak je to možné.

Abychom obdrželi I/O kontrolu pevné pozice, `TriggCheckIO` kompenzuje zpoždění v kontrolním systému (zpoždění mezi servem a robotem).

Definovaná data se používají pro implementaci do jedné nebo více následných instrukcích `TriggL`, `TriggC`, nebo `TriggJ` .

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggCheckIO`:

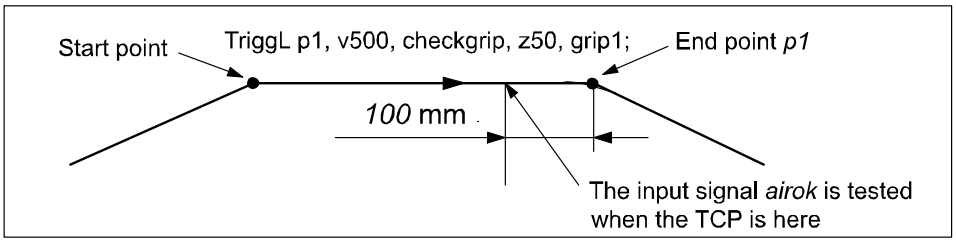
Viz také *Další příklady na str 807*.

Příklad 1

```
VAR triggdata checkgrip;
VAR intnum intno1;

PROC main()
  CONNECT intno1 WITH trap1;
  TriggCheckIO checkgrip, 100, airok, EQ, 1, intno1;

  TriggL p1, v500, checkgrip, z50, grip1;
```

Digitální vstupní signál `airok` je kontrolován, jestli má hodnotu `1`, když TCP je `100` mm před bodem `p1`. Jestliže je nastaven, pokračuje normální vykonávání programu. Jestliže není nastaven, je provedena rutina přerušení `trap1`.

Obrázek ukazuje příklad kontroly I/O pevné pozice.



xx0500002254

Argumenty

```
TriggCheckIO TriggData Distance [\Start] | [\Time] Signal Relation
     CheckValue | CheckDvalue [\StopMove] Interrupt [\Inhib]
     [\Mode]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`TriggData`

Datový typ: `triggdata`

Proměnná pro uložení `triggdata` vrácených z této instrukce. Tato `triggdata` jsou potom použita v následných instrukcích `TriggL`, `TriggC` nebo `TriggJ`.

`Distance`

Datový typ: `num`

Definuje pozici na dráze, kde by mělo dojít ke kontrole I/O.

Určeno jako vzdálenost v mm (kladná hodnota) od koncového bodu dráhy pohybu (použitelné, jestliže není nastaven argument `\Start` nebo `\Time`).

Další podrobnosti viz *Vykonávání programu na str 806*.

`[ \Start ]`

Datový typ: `switch`

Používá se, když vzdálenost pro argument `Distance` začíná v bodě začátku pohybu namísto v koncovém bodě.

`[ \Time ]`

Datový typ: `switch`

Používá se, když hodnota určená pro argument `Distance` je ve skutečnosti čas v sekundách (0,5 s) (kladná hodnota) namísto vzdálenosti.

I/O pevné pozice v čase se může použít pouze pro krátké časy (< 0,5 s), předtím, než robot dosáhne koncového bodu instrukce. Více podrobností najdete v sekci *Omezení*

`Signal`

Datový typ: `signalxx`

Jméno signálu, který bude testován. Může to být kterýkoliv typ IO signálu.

`Relation`

Datový typ: `opnum`

Definuje, jak porovnávat aktuální hodnotu signálu a tím, který byl definován argumentem `CheckValue`. Viz datový typ `opnum`, kde je seznam používaných předdefinovaných konstant.

`CheckValue`

Datový typ: `num`

Hodnota, se kterou bude porovnávána aktuální hodnota vstupního nebo výstupního signálu (v rámci povoleného rozsahu pro aktuální signál). Jestliže signál je digitálním signálem, musí to být hodnota celého čísla.

Jestliže signál je digitálním skupinovým signálem, povolená hodnota je závislá na počtu signálů ve skupině. Max hodnota, kterou je možné použít v argumentu `CheckValue`, je 8388608, a to je hodnota, kterou může mít 23bitová skupina digitálních signálů jako max hodnotu (viz rozsahy pro `num`).).

`CheckDvalue`

Datový typ: `dnum`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.290 TriggCheckIO - Definuje kontrolu IO v pevné pozici
RobotWare - OS
Pokračování

Hodnota, se kterou bude porovnávána aktuální hodnota vstupního nebo výstupního signálu (v rámci povoleného rozsahu pro aktuální signál). Jestliže signál je digitálním signálem, musí to být hodnota celého čísla.

Jestliže signál je digitálním skupinovým signálem, povolená hodnota je závislá na počtu signálů ve skupině. Max množství signálových bitů, které může skupina digitálních signálů mít, je 32. S proměnnou `dnum` je možné pokrýt rozsah hodnoty 0-4294967295, což je hodnotový rozsah, jaký může 32bitový digitální signál mít.

`[ \StopMove ]`

Datový typ: `switch`

Určuje, že když podmínka není splněna, robot se zastaví na dráze tak rychle, jak je to možné před provedením rutiny přerušení.

`Interrupt`

Datový typ: `intnum`

Proměnná použitá k identifikaci rutiny, která se bude provádět.

`[ \Inhib ]`

Inhibit

Datový typ: `bool`

Jméno příznaku perzistentní proměnné pro zpomalení vykonávání rutiny přerušení.

Jestliže se používá tento volitelný argument a aktuální hodnota určeného příznaku je TRUE v čase pozice pro kontrolu I/O, kontrola nebude provedena.

`[ \Mode ]`

Datový typ: `triggmode`

Používá se pro určení odlišných režimů činností při definování spouštěčů.

Vykonávání programu

Při provádění instrukce `TriggCheckIO` je podmínka spouštěče uložena do určené proměnné pro argument `TriggData`.

Později, když se provádí jedna z instrukcí `TriggL`, `TriggC` nebo `TriggJ`, následující jsou použitelné s ohledem na definice v `TriggCheckIO`:

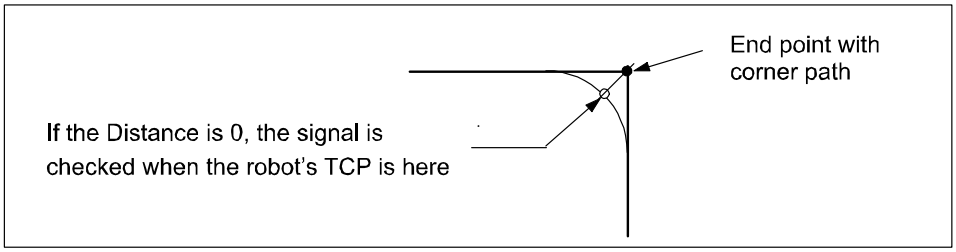
Tabulka popisuje vzdálenost určenou v argumentu `Distance`:

Lineární pohyb	Přímá vzdálenosti
Kruhový pohyb	Délka kruhového oblouku
Nelineární pohyb	Přibližná délka oblouku podél dráhy (abychom dostali adekvátní přesnost, vzdálenost by neměla překročit polovinu délky oblouku).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Obrázek ukazuje I/O kontrolu pevné pozice na rohové dráze.



xx0500002256

I/O kontrola pevné pozice bude provedena, když je přejet počáteční bod (koncový bod), jestliže určená vzdálenost od koncového bodu (počátečního bodu) není v rámci délky pohybu aktuální instrukce (`TriggL...`).

Když TCP robotu je na určeném místě na dráze, systém provede následující I/O kontrolu:

- Přečíst hodnotu I/O signálu.
- Porovnat přečtenou hodnotu s `CheckValue` podle určeného `Relation`.
- Jestliže porovnání je TRUE nebude už provedeno nic dalšího.
- Jestliže porovnání je FALSE bude provedeno následující:
- Jestliže volitelný parametr `\StopMove` je přítomen, potom je robot zastaven na dráze tak rychle, jak je to možné.
- Generovat a vykonat určenou TRAP rutinu.

Další příklady

Více příkladů jak používat instrukci `TriggCheckIO` je názorně uvedeno dole.

Příklad 1

```
VAR triggdata checkgate;
VAR intnum gateclosed;

PROC main()
  CONNECT gateclosed WITH waitgate;
  TriggCheckIO checkgate,150, gatedi, EQ, 1 \StopMove, gateclosed;
  TriggL p1, v600, checkgate, z50, grip1;
  ...
TRAP waitgate
  ! Block movement
  StopMove;
  ! log some information
  ...
  ! Wait until signal is set
  WaitDI gatedi,1;
  ! Unlock block, and resume movement
  StartMove;
ENDTRAP
```

Brána pro další operaci pracovního kusu je zkontrolována kvůli otevření (digitální vstupní signál `gatedi` je zkontrolován, jestli má hodnotu `1`), když TCP je `150`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.290 TriggCheckIO - Definuje kontrolu IO v pevné pozici
RobotWare - OS
Pokračování

mm před bodem p1. Jestliže je otevřen, robot se posune dál na p1 a pokračuje. Jestliže není otevřen, robot je zastaven na dráze a rutina přerušení waitgate je provedena. Toto přerušení blokuje další pohyby, zapíše některé informace a typicky čeká, až podmínky budou OK, aby mohla být vykonána instrukce StartMove pro restart přerušené dráhy.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_AO_LIM

jestliže naprogramovaný argument CheckValue nebo CheckDvalue pro určený analogový výstupní signál Signal je mimo limity.

ERR_GO_LIM

jestliže naprogramovaný argument CheckValue nebo CheckDvalue pro určený digitální skupinový výstupní signál Signal je mimo limity.

ERR_NO_ALIASIO_DEF

jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.

Omezení

I/O kontroly se vzdáleností (bez argumentu \Time) jsou určeny pro průjezdné body (rohová dráha). I/O kontroly se vzdáleností, pomocí stop bodů, mají horší přesnost,než je uvedeno dole.

I/O kontroly s časem (s argumentem \Time) jsou určeny pro stop body. I/O kontroly s časem, pomocí průjezdných bodů, mají horší přesnost,než je uvedeno dole.

I/O kontroly s časem mohou být určeny pouze od koncového bodu pohybu. Tento čas nemůže překročit aktuální brzdný čas robotu, který je max. 0,5 sek. (typické hodnoty při rychlosti 500 mm/s pro IRB2400 150 ms a pro IRB6400 250 ms). Jestliže určený čas je delší než aktuální brzdný čas, potom bude I/O kontrola přesto generována, ale nikoliv před zahájením brzdění (později než je určeno). Celkový čas pohybu pro aktuální pohyb může být použit během malých a rychlých pohybů.

Typické hodnoty absolutní přesnosti pro testování digitálních vstupů +/- 5 ms. Typické hodnoty opakované přesnosti pro testování digitálních vstupů +/- 2 ms.

Syntaxe

```
TriggCheckIO
  [ TriggData ':=' ] < variable (VAR) of triggdata> ','
  [ Distance' :=' ] < expression (IN) of num>
  [ '\' Start ] | [ '\' Time ] ','
  [ Signal ':=' ] < variable (VAR) of anytype> ','
  [ Relation' :=' ] < expression (IN) of opnum> ','
  [ CheckValue' :=' ] < expression (IN) of num>
  | [ CheckDvalue' :=' ] < expression (IN) of dnum>
  [ '\' StopMove] ','
  [ Interrupt' :=' ] < variable(VAR) of intnum>
  [ '\' Inhib' :=' < persistent (PERS) of bool> ]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
[ Mode' :=' ] < expression (IN) of triggmode> ';'
```

Související informace

Pro informace o	Viz
Použití spouštěčů	*TriggL - Lineární pohyby robotu s událostmi na str 839* *TriggC - Kruhový pohyb robotu s událostmi na str 796* *TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Definice I/O události pozice-čas	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814*
Definice přerušení se vztahem k pozici	*TriggInt - Definuje přerušení se vztahem k pozici na str 820*
Uložení trigg dat	*triggdata - Polohovací události, trigg na str 1618* *triggmode - Režim činnosti trigg na str 1624*
Definice operátorů srovnání	*opnum - Srovnávací operátor na str 1541*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.291 TriggDataCopy - Kopírovat obsah do proměnné triggdata

Použití

`TriggDataCopy` se používá ke kopírování obsahu do proměnné `triggdata`.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggDataCopy`.

Příklad 1

```
VAR triggdata trigg_array{25};
...
PROC MyTriggProcL(robtarget myrobt, \VAR triggdata T1 \VAR triggdata
     T2 \VAR triggdata T3)
  VAR num triggcnt:=2;
  ! Reset entire trigg_array array before using it
  FOR i FROM 1 TO 25 DO
    TriggDataReset trigg_array{i};
  ENDFOR
  TriggEquip trigg_array{1}, 10 \Start, 0 \DOp:=do1, SetValue:=1;
  TriggEquip trigg_array{2}, 40 \Start, 0 \DOp:=do2, SetValue:=1;
  ! Check if optional argument is present,
  ! and if any trigger condition has been setup in T1
  IF Present(T1) AND TriggDataValid(T1) THEN
    ! Copy actual trigger condition to trigg_array
    TriggDataCopy T1, trigg_array{triggcnt};
    Incr triggcnt;
  ENDIF
  IF Present(T2) AND TriggDataValid(T2) THEN
    Incr triggcnt;
    TriggDataCopy T2, trigg_array{triggcnt};
  ENDIF
  IF Present(T3) AND TriggDataValid(T3) THEN
    Incr triggcnt;
    TriggDataCopy T3, trigg_array{triggcnt};
  ENDIF
  TriggL p1, v500, trigg_array, z30, tool2;
  ...
```

Procedura `MyTriggProcL` nahoře používá instrukci `TriggDataCopy` pro kopírování volitelných argumentů `triggdata` na správné místo v poli `triggdata`, které je použito v instrukci `TriggL`.

Argumenty

```
TriggDataCopy Source Destination
```

`Source`

Datový typ: `triggdata`

Proměnná `triggdata`, ze které se bude kopírovat.

`Destination`

Datový typ: `triggdata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Proměnná `triggdata`, do které se bude kopírovat.

Vykonávání programu

Instrukce `TriggDataCopy` se používá pro kopírování dat z jedné proměnné `triggdata` do druhé proměnné `triggdata`. Tato instrukce může být výhodná při práci s proměnnými pole `triggdata`.

Syntaxe

```
TriggDataCopy
  [Source ':=' ] < variable (VAR) of triggdata > ','
  [Destination ':=' ] < variable (VAR) of triggdata > ';'
```

Související informace

Pro informace o	Viz
Lineární pohyb se spouštěči	*TriggL - Lineární pohyby robotu s událostmi na str 839*
Pohyb spoje se spouštěči	*TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Kruhový pohyb se spouštěči	*TriggC - Kruhový pohyb robotu s událostmi na str 796*
Definice spouštěčů	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814* *TriggInt - Definuje přerušení se vztahem k pozici na str 820* *TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804* *TriggRampAO - Definovat událost rampy AO pevné pozice na dráze na str 861* *TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času na str 868*
Manipulace s `triggdata`	*triggdata - Polohovací události, trigg na str 1618* *TriggDataReset - Resetovat obsah v proměnné triggdata na str 812* *TriggDataValid - Zkontrolujte, jestli obsah proměnné triggdata je platný na str 1374*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.292 TriggDataReset - Resetovat obsah v proměnné triggdata

Použití

`TriggDataReset` se používá k resetu obsahu v proměnné `triggdata`.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggDataReset`.

Příklad 1

```
VAR triggdata trigg_array{25};
...
PROC MyTriggProcL(robtarget myrobt, \VAR triggdata T1 \VAR triggdata
     T2 \VAR triggdata T3)
  VAR num triggcnt:=2;
  ! Reset entire trigg_array array before using it
  FOR i FROM 1 TO 25 DO
    TriggDataReset trigg_array{i};
  ENDFOR
  TriggEquip trigg_array{1}, 10 \Start, 0 \DOp:=do1, SetValue:=1;
  TriggEquip trigg_array{2}, 40 \Start, 0 \DOp:=do2, SetValue:=1;
  ! Check if optional argument is present,
  ! and if any trigger condition has been setup in T1
  IF Present(T1) AND TriggDataValid(T1) THEN
    ! Copy actual trigger condition to trigg_array
    TriggDataCopy trigg_array{triggcnt}, T1;
    Incr triggcnt;
  ENDIF
  IF Present(T2) AND TriggDataValid(T2) THEN
    Incr triggcnt;
    TriggDataCopy trigg_array{triggcnt}, T2;
  ENDIF
  IF Present(T3) AND TriggDataValid(T3) THEN
    Incr triggcnt;
    TriggDataCopy trigg_array{triggcnt}, T3;
  ENDIF
  TriggL p1, v500, trigg_array, z30, tool2;
  ...
```

Procedura `MyTriggProcL` nahoře používá instrukci `TriggDataReset` k resetu pole `triggdata` předtím, než je použito.

Argumenty

```
TriggDataReset TriggData
```

`TriggData`

Datový typ: `triggdata`

Proměnná `triggdata` pro resetování.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Instrukce `TriggDataReset` se používá pro odstranění každé podmínky spouštěče předtím použité v proměnné `triggdata`. Tato instrukce může být výhodná při práci s proměnnými pole `triggdata`.

Syntaxe

```
TriggDataReset
  [TriggData ':=' ] < variable (VAR) of triggdata > ';'
```

Související informace

Pro informace o	Viz
Lineární pohyb se spouštěči	*TriggL - Lineární pohyby robotu s událostmi na str 839*
Pohyb spoje se spouštěči	*TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Kruhový pohyb se spouštěči	*TriggC - Kruhový pohyb robotu s událostmi na str 796*
Definice spouštěčů	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814* *TriggInt - Definuje přerušení se vztahem k pozici na str 820* *TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804* *TriggRampAO - Definovat událost rampy AO pevné pozice na dráze na str 861* *TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času na str 868*
Manipulace s `triggdata`	*triggdata - Polohovací události, trigg na str 1618* *TriggDataCopy - Kopírovat obsah do proměnné triggdata na str 810* *TriggDataValid - Zkontrolujte, jestli obsah proměnné triggdata je platný na str 1374*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.293 TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze

Použití

`TriggEquip` (*Trigg Equipment*) se používá k definování podmínek a činností pro nastavení digitálního, skupiny digitálních nebo analogového výstupního signálu na pevné pozici podél dráhy pohybu robotu s možností provedení kompenzace času kvůli zpoždění v externím zařízení.

`TriggIO` (nikoliv `TriggEquip`) by se mělo vždy používat, jestliže existuje potřeba dobré přesnosti nastavení I/O poblíž stop bodu.

Definovaná data se používají pro implementaci do jedné nebo více následných instrukcích `TriggL`, `TriggC`, nebo `TriggJ` .

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggEquip`:

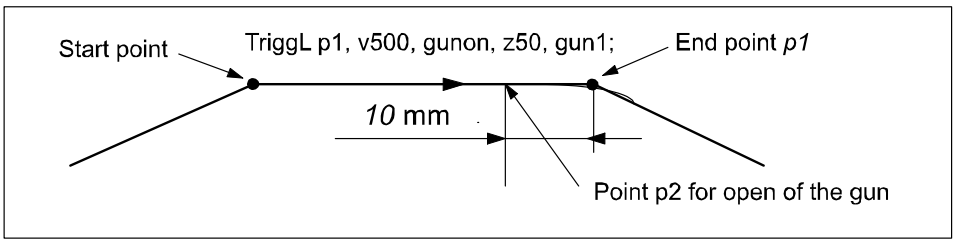
Viz také *Další příklady na str 818*.

Příklad 1

```
VAR triggdata gunon;
...
TriggEquip gunon, 10, 0.1 \DOp:=gun, 1;
TriggL p1, v500, gunon, z50, gun1;
```

Nástroj `gun1` se začíná otevírat, když jeho TCP je `0,1` s před domnělým bodem `p2` (`10` mm před bodem `p1`). Pistole je zcela otevřena, když TCP dosáhne bodu `p2`.

Obrázek ukazuje příklad časové I/O události pevné pozice.



xx0500002260

Argumenty

```
TriggEquip TriggData Distance [\Start] | [\Next] EquipLag [\DOp]
     | [\GOp]| [\AOp] | [\ProcID] SetValue | SetDvalue [\Inhib]
     [\InhibSetValue] [\Mode]
```

`TriggData`

Datový typ: `triggdata`

Proměnná pro uložení `triggdata` vrácených z této instrukce. Tato `triggdata` jsou potom použita v následných instrukcích `TriggL`, `TriggC` nebo `TriggJ`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Distance`

Datový typ: `num`

Definuje pozici na dráze, kde by mělo dojít k události I/O zařízení.

Určeno jako vzdálenost v mm (kladná hodnota) od koncového bodu dráhy pohybu k počátečnímu bodu (použitelné, jestliže nejsou nastaveny argumenty `\Start` a `\Next`).

Další podrobnosti viz *Vykonávání programu na str 817*.

`[ \Start ]`

Datový typ: `switch`

Používá se, když vzdálenost pro argument `Distance` začíná v bodě začátku pohybu namísto v koncovém bodě.

`[ \Next ]`

Datový typ: `switch`

Používá se, když vzdálenost pro argument `Distance` je dopředu k dalšímu naprogramovanému bodu. Jestliže `Distance` je delší než vzdálenost k příštímu jemnému bodu, událost bude provedena u jemného bodu.

`EquipLag`

Equipment Lag

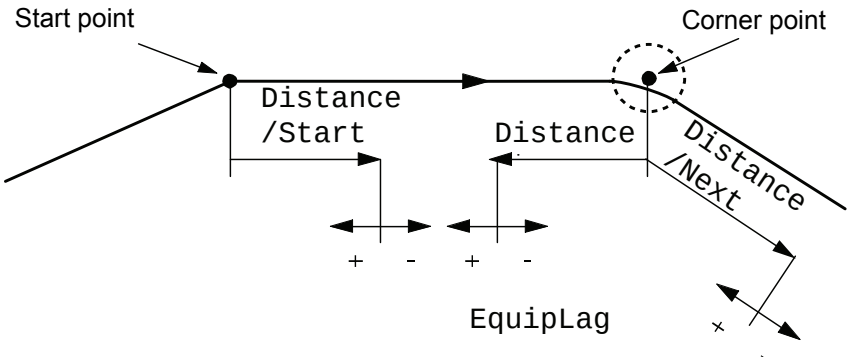
Datový typ: `num`

Určit zpoždění pro externí zařízení v sek.

Pro kompenzaci zpoždění externího zařízení použijte kladnou hodnotu argumentu. Kladná hodnota argumentu znamená, že I/O signál je nastaven systémem robotu na určený čas, předtím, než TCP fyzicky dosáhne určené vzdálenosti ve vztahu k začátku pohybu nebo koncovému bodu.

Záporná hodnota argumentu znamená, že I/O signál je nastaven systémem robotu na určený čas, po kterém TCP fyzicky přejde určenou vzdálenost ve vztahu k začátku pohybu nebo koncovému bodu.

Obrázek ukazuje použití argumentu `EquipLag`.



xx0500002262

`[ \DOp ]`

Digital Output

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.293 TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze
RobotWare - OS
Pokračování

Datový typ: `signaldo`

Jméno signálu, když digitální výstupní signál bude změněn.

`[ \GOp ]`

Group Output

Datový typ: `signalgo`

Jméno signálu, když skupina digitálních výstupních signál bude změněna.

`[ \AOp ]`

Analog Output

Datový typ: `signalao`

Jméno signálu, když analogový výstupní signál bude změněn.

`[ \ProcID ]`

Process Identity

Datový typ: `num`

Není implementováno pro zákaznické použití.

(Identita IPM procesu pro přijetí události. Selektor je určen v argumentu `SetValue`.)

`SetValue`

Datový typ: `num`

Požadovaná hodnota signálu (v rámci povoleného rozpětí pro aktuální signál). Jestliže signál je digitálním signálem, musí to být hodnota celého čísla. Jestliže signál je digitálním skupinovým signálem, povolená hodnota je závislá na počtu signálů ve skupině. Max hodnota, kterou je možné použít v argumentu `SetValue`, je 8388608, a to je hodnota, kterou může mít 23bitová skupina digitálních signálů jako max hodnotu (viz rozsahy pro `num`).

`SetDvalue`

Datový typ: `dnum`

Požadovaná hodnota signálu (v rámci povoleného rozpětí pro aktuální signál). Jestliže signál je digitálním signálem, musí to být hodnota celého čísla. Jestliže signál je digitálním skupinovým signálem, povolená hodnota je závislá na počtu signálů ve skupině. Max množství signálových bitů, které může skupina digitálních signálů mít, je 32. S proměnnou `dnum` je možné pokrýt rozsah hodnoty 0-4294967295, což je hodnotový rozsah, jaký může 32bitový digitální signál mít.

`[ \Inhib ]`

Inhibit

Datový typ: `bool`

Jméno příznaku perzistentní proměnné pro zpomalení nastavení signálu v čase běhu.

Jestliže je použit tento volitelný argument a aktuální hodnota určeného příznaku je TRUE na pozici-času pro nastavení signálu, potom bude určený signál (`DOp`, `GOp` nebo `AOp`) nastaven na 0 namísto určené hodnoty.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[ \InhibSetValue ]`

InhibitSetValue

Datový typ: `bool, num or dnum`

Jméno perzistentní proměnné datového typu `bool`, `num` **nebo** `dnum` **nebo kterýkoliv alias těchto tří základnových datových typů.**

Tento volitelný argument může být použit pouze společně s volitelným argumentem `Inhib`.

Jestliže je použit tento volitelný argument a hodnota příznaku perzistentní proměnné použité ve volitelném argumentu `Inhib` **je TRUE u pozice-čas pro nastavení signálu, hodnota perzistentní proměnné použité ve volitelném argumentu** `InhibSetValue` **je přečtena a hodnota je použita pro nastavení signálu** `DOp`, `GOp` **nebo** `AOp`.

Jestliže používáme booleánskou perzistentní proměnnou, hodnota TRUE je přeložena na hodnotu 1 a FALSE je přeloženo na hodnotu 0.

`[ \Mode ]`

Datový typ: `triggmode`

Používá se pro určení odlišných režimů činností při definování spouštěčů.

Vykonávání programu

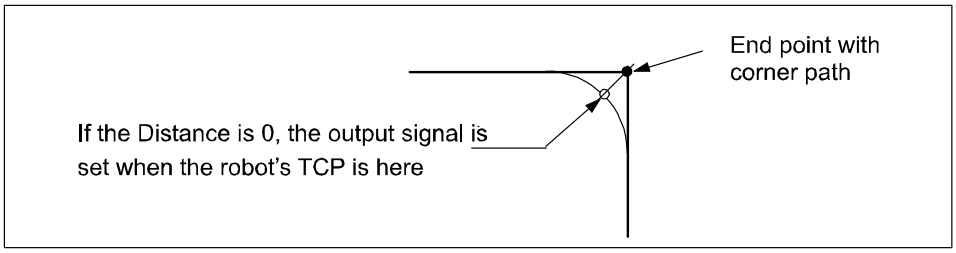
Při provádění instrukce `TriggEquip` **je podmínka spouštěče uložena do určené proměnné pro argument** `TriggData`.

Později, když se provádí jedna z instrukcí `TriggL`, `TriggC` **nebo** `TriggJ`, **následující jsou použitelné s ohledem na definice v** `TriggEquip`:

Tabulka popisuje vzdálenost určenou v argumentu `Distance`:

Lineární pohyb	Přímá vzdálenosti
Kruhový pohyb	Délka kruhového oblouku
Nelineární pohyb	Přibližná délka oblouku podél dráhy (abychom dostali adekvátní přesnost, vzdálenost by neměla překročit polovinu délky oblouku).

Obrázek ukazuje pevnou I/O pozice-čas na rohové dráze.



xx0500002263

Událost ve vztahu pozice-čas bude generována, když je přejet počáteční bod (koncový bod), jestliže určená vzdálenost od koncového bodu (počátečního bodu) není v rámci délky pohybu aktuální instrukce (`TriggL...`**). S použitím argumentu** `EquipLag` **se záporným časem (prodleva) může být I/O signál nastaven po koncovém bodu.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.293 TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze
RobotWare - OS
Pokračování

Další příklady

Více příkladů jak používat instrukci `TriggEquip` je názorně uvedeno dole.

Příklad 1

```
VAR triggdata glueflow;
...
TriggEquip glueflow, 1 \Start, 0.05 \AOp:=glue, 5.3;
MoveJ p1, v1000, z50, tool1;
TriggL p2, v500, glueflow, z50, tool1;
```

Analogový výstupní signál `glue` je nastaven na hodnotu `5.3` , když TCP přejede bod umístěný `1` mm za počátečním bodem `p1` s kompenzací pro zpoždění zařízení `0.05` s.

Příklad 2

```
...
TriggL p3, v500, glueflow, z50, tool1;
```

Analogový výstupní signál `glue` je nastaven ještě jednou na hodnotu `5.3` , když TCP přejede bod umístěný `1` mm za počátečním bodem `p2`.

Řešení chyb

Jestliže naprogramovaný argument `SetValue` pro určený analogový výstupní signál `AOp` je mimo limit, potom je systémová proměnná `ERRNO` nastavena na `ERR_AO_LIM`. Tato chyba může být ošetřena v chybovém handleru.

Jestliže naprogramovaný argument `SetValue` nebo `SetDvalue` pro určený digitální skupinový výstupní signál `GOp` je mimo limit, potom je systémová proměnná `ERRNO` nastavena na `ERR_GO_LIM`. Tato chyba může být ošetřena v chybovém handleru.

Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`, potom je systémová proměnná `ERRNO` nastavena na `ERR_NO_ALIASIO_DEF`. Tato chyba může být ošetřena v chybovém handleru.

Omezení

I/O události se vzdáleností jsou určeny pro průjezdné body (rohová dráha). I/O události se vzdáleností, pomocí stop bodů, mají horší přesnost, než je uvedeno dole.

S ohledem na přesnost u I/O událostí se vzdáleností a pomocí průjezdných bodů, následující je použitelné při nastavování digitálního výstupu v určené vzdálenosti od počátečního bodu nebo koncového bodu v instrukci `TriggL` nebo `TriggC`:

- Přesnost určená dole je platná pro kladný parametr `EquipLag` parameter `<` 40 ms, ekvivalent ke zpoždění v servu robotu (beze změny systémového parametru `Event Preset Time`). Zpoždění může kolísat mezi různými typy robotů. Např. je nižší u IRB140.
- Přesnost určená dole je platná pro kladný parametr `EquipLag` `<` konfigurovaný `Event Preset Time` (systémový parametr).
- Přesnost určená dole není platná pro kladný parametr `EquipLag` `>` konfigurovaný `Event Preset Time` (systémový parametr). V tomto případě

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

je použita přibližná metoda, ve které dynamická omezení robotu nejsou brána v úvahu. `SingArea \Wrist` se musí použít k dosažení přijatelné přesnosti.

- Přesnost určená dole je platná pro záporný `EquipLag`.

Typické hodnoty absolutní přesnosti pro sadu digitálních výstupů +/- 5 ms.

Typické hodnoty opakované přesnosti pro sadu digitálních výstupů +/- 2 ms.

Syntaxe

```
TriggEquip
  [ TriggData ':=' ] < variable (VAR) of triggdata> ','
  [ Distance' :=' ] < expression (IN) of num>
  [ '\' Start ]
  | [ '\' Next ] ','
  [ EquipLag' :=' ] < expression (IN) of num>
  [ '\' DOp' :=' < variable (VAR) of signaldo> ]
  | [ '\' GOp' :=' < variable (VAR) of signalgo> ]
  | [ '\' AOp' :=' < variable (VAR) of signalao> ]
  | [ '\' ProcID' :=' < expression (IN) of num> ] ','
  [ SetValue' :=' ] < expression (IN) of num>
  | [ SetDvalue' :=' ] < expression (IN) of dnum> ','
  [ '\' Inhib' :=' < persistent (PERS) of bool> ]
  [ '\' InhibSetValue' :=' < persistent (PERS) of anytype> ]
  [ Mode' :=' ] < expression (IN) of triggmode> ';'
```

Související informace

Pro informace o	Viz
Použití spouštěčů	*TriggL - Lineární pohyby robotu s událostmi na str 839* *TriggC - Kruhový pohyb robotu s událostmi na str 796* *TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Definice ostatních trigg	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggInt - Definuje přerušení se vztahem k pozici na str 820*
Definovat kontrolu I/O na pevné pozici	*TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804*
Uložení trigg dat	*triggdata - Polohovací události, trigg na str 1618* *triggmode - Režim činnosti trigg na str 1624*
Sada I/O	*SetDO - Mění hodnotu digitálního výstupního signálu na str 629* *SetGO - Mění hodnotu skupiny digitálních výstupních signálů na str 631* *SetAO - Mění hodnotu analogového výstupního signálu na str 620*
Konfigurace Event preset time	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.294 TriggInt - Definuje přerušení se vztahem k pozici

Použití

`TriggInt` se používá pro definování podmínek a činností pro provádění rutiny přerušení na určené pozici na dráze pohybu robotu.

Definovaná data se používají pro implementaci do jedné nebo více následných instrukcích `TriggL`, `TriggC`, nebo `TriggJ` .

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

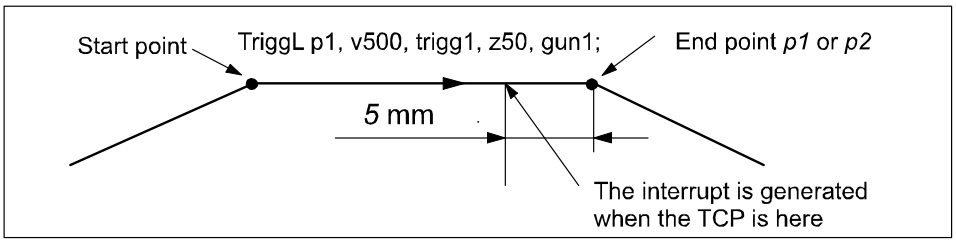
Následující příklad názorně ukazuje instrukci `TriggInt`:

Příklad 1

```
VAR intnum intno1;
VAR triggdata trigg1;
...
PROC main()
  CONNECT intno1 WITH trap1;
  TriggInt trigg1, 5, intno1;
  ...
  TriggL p1, v500, trigg1, z50, gun1;
  TriggL p2, v500, trigg1, z50, gun1;
  ...
  IDelete intno1;
```

Rutina proměnné `trap1` je prováděna, když TCP je v pozici `5`mm před bodem `p1` nebo `p2`.

Obrázek ukazuje příklad přerušení ve vztahu k pozici.



xx0500002251

Argumenty

```
TriggInt TriggData Distance [\Start] | [\Time] Interrupt [\Inhib]
     [\Mode]
```

`TriggData`

Datový typ: `triggdata`

Proměnná pro uložení `triggdata` vrácených z této instrukce. Tato `triggdata` jsou potom použita v následných instrukcích `TriggL`, `TriggC` nebo `TriggJ`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Distance`

Datový typ: `num`

Definuje pozici na dráze, kde by mělo být generováno přerušení.

Určeno jako vzdálenost v mm (kladná hodnota) od koncového bodu dráhy pohybu (použitelné, jestliže není nastaven argument `\Start` nebo `\Time`).

Další podrobnosti viz *Vykonávání programu na str 821*.

`[ \Start ]`

Datový typ: `switch`

Používá se, když vzdálenost pro argument `Distance` začíná v bodě začátku pohybu namísto v koncovém bodě.

`[ \Time ]`

Datový typ: `switch`

Používá se, když hodnota určená pro argument `Distance` je ve skutečnosti čas v sekundách (0,5 s) (kladná hodnota) namísto vzdálenosti.

Přerušení se vztahem k pozici v čase se mohou použít pouze pro krátké časy (< 0,5 s), předtím, než robot dosáhne koncového bodu instrukce. Více podrobností najdete v sekci *Omezení*

`Interrupt`

Datový typ: `intnum`

Proměnná použitá pro identifikaci přerušení.

`[ \Inhib ]`

Inhibit

Datový typ: `bool`

Jméno příznaku perzistentní proměnné pro zpomalení vykonávání rutiny přerušení.

Jestliže se používá tento volitelný argument a aktuální hodnota určeného příznaku je TRUE v pozici-čase pro vykonání přerušení, přerušení nebude provedeno.

`[ \Mode ]`

Datový typ: `triggmode`

Používá se pro určení odlišných režimů činností při definování spouštěčů.

Vykonávání programu

Při provádění instrukce `TriggInt` jsou data uložena do určené proměnné pro argument `TriggData` a přerušení, které je určeno v proměnné pro argument `Interrupt`, je aktivováno.

Později, když se provádí jedna z instrukcí `TriggL`, `TriggC` nebo `TriggJ`, následující jsou použitelné s ohledem na definice v `TriggInt`:

Tabulka popisuje vzdálenost určenou v argumentu `Distance`:

Lineární pohyb	Přímá vzdálenosti
Kruhový pohyb	Délka kruhového oblouku

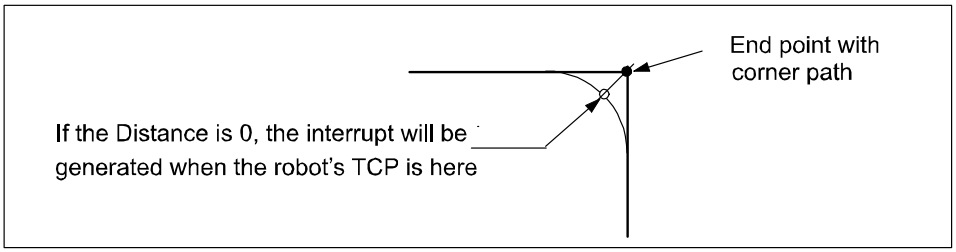
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.294 TriggInt - Definuje přerušení se vztahem k pozici
RobotWare - OS
Pokračování

Nelineární pohyb	Přibližná délka oblouku podél dráhy (abychom dostali adekvátní přesnost, vzdálenost by neměla překročit polovinu délky oblouku).

Obrázek ukazuje přerušení ve vztahu k pozici na rohové dráze.



xx0500002253

Přerušení ve vztahu k pozici bude generováno, když je přejet počáteční bod (koncový bod), jestliže určená vzdálenost od koncového bodu (počátečního bodu) není v rámci délky pohybu aktuální instrukce (`TriggL...`).

Přerušení je považováno za bezpečné přerušení. Bezpečné přerušení nemůže být uloženo ke spánku s instrukcí `ISleep`. Událost bezpečného přerušení bude umístěna do fronty při zastavení programu a krokovém vykonávání, a při spuštění znovu v plynulém režimu bude přerušení vykonáno. Jediným časem, kdy bude bezpečné přerušení vyhozeno, je zaplnění fronty přerušení. Potom bude hlášena chyba. Přerušení nepřežije reset programu, např. PP na main.

Další příklady

Více příkladů jak používat instrukci `TriggInt` je názorně uvedeno dole.

Příklad 1

Tento příklad popisuje programování instrukcí, které se vzájemně ovlivňují při generování přerušení ve vztahu k pozici:

```
VAR intnum intno2;
VAR triggdata trigg2;
```

- Deklarace proměnných `intno2` a `trigg2` (by neměla být iniciována).

```
CONNECT intno2 WITH trap2;
```

- Přidělení čísel přerušení, která jsou uložena v proměnné `intno2`.
- Číslo přerušení je propojeno s rutinou přerušení `trap2`.

```
TriggInt trigg2, 0, intno2;
```

- Číslo přerušení v proměnné `intno2` je opatřeno příznakem jako použité.
- Přerušení je aktivováno.
- Definované spouštěcí podmínky a čísla přerušení jsou uložena do proměnné `trigg2`

```
TriggL p1, v500, trigg2, z50, gun1;
```

- Robot je přesunut do bodu `p1`.
- Když TCP dosáhne bodu `p1`, přerušení je generováno a rutina přerušení `trap2` je provedena.

```
TriggL p2, v500, trigg2, z50, gun1;
```

- Robot je přesunut do bodu `p2`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

- Když TCP dosáhne bodu `p2`, přerušení je generováno a rutina přerušení `trap2` je provedena ještě jednou.

```
IDelete intno2;
```

- Číslo přerušení v proměnné `intno2` je dealokováno.

Omezení

Události přerušení se vzdáleností (bez argumentu `\Time`) jsou určeny pro průjezdné body (rohová dráha). Události přerušení se vzdáleností, pomocí stop bodů, mají horší přesnost, než je uvedeno dole.

Události přerušení s časem (s argumentem `\Time`) jsou určeny pro stop body. Události přerušení s časem, pomocí průjezdných bodů, mají horší přesnost než je uvedeno dole. Události I/O s časem mohou být určeny pouze od koncového bodu pohybu. Tento čas nemůže překročit aktuální brzdný čas robotu, který je max. 0,5 sek. (typické hodnoty při rychlosti 500 mm/s pro IRB2400 150 ms a pro IRB6400 250 ms). Jestliže určený čas je delší než aktuální brzdný čas, potom bude událost přesto generována, ale nikoliv před zahájením brzdění (později než je určeno). Celkový čas pohybu pro aktuální pohyb může být použit během malých a rychlých pohybů.

Typické hodnoty absolutní přesnosti pro generování přerušení +/- 5 ms. Typické hodnoty opakované přesnosti pro generování přerušení +/- 2 ms. Normálně je zde prodleva 2 až 30 ms mezi generováním přerušení a odezvou podle typu prováděného pohybu v okamžiku přerušení. (Ref. k *Referenční příručka RAPID - Přehled RAPID*, sekce *Základní vlastnosti - Přerušení*).

Abyste dosáhli nejvyšší přesnosti při nastavování pevné polohy podél dráhy robotu, použijte instrukce `TriggIO` nebo `TriggEquip` jako preferenci k instrukcím `TriggInt` ws `SetDO/SetGO/SetAO` v rutině přerušení.

Syntaxe

```
TriggInt
  [ TriggData ':=' ] < variable (VAR) of triggdata> ','
  [ Distance' :=' ] < expression (IN) of num>
  [ '\' Start ] | [ '\' Time ] ','
  [ Interrupt' :=' ] < variable (VAR) of intnum>
  [ '\' Inhib' :=' < persistent (PERS) of bool> ]
  [ Mode' :=' ] < expression (IN) of triggmode> ';'
```

Související informace

Pro informace o	Viz
Použití spouštěčů	*TriggL - Lineární pohyby robotu s událostmi na str 839* *TriggC - Kruhový pohyb robotu s událostmi na str 796* *TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Definice I/O opravného kódu pozice	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.294 TriggInt - Definuje přerušení se vztahem k pozici

RobotWare - OS

Pokračování

Pro informace o	Viz
Definovat kontrolu I/O na pevné pozici	*TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804*
Uložení trigg dat	*triggdata - Polohovací události, trigg na str 1618* *triggmode - Režim činnosti trigg na str 1624*
Přerušení	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.295 TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu

Použití

`TriggIO` se používá pro definování podmínek a činností pro nastavení digitálního, skupiny digitálních nebo analogového výstupního signálu na pevné pozici podél dráhy robotu.

`TriggIO` (nikoliv `TriggEquip`) by se mělo vždy používat, jestliže existuje potřeba dobré přesnosti nastavení I/O poblíž stop bodu.

Aby bylo dosaženo I/O události pevné pozice, `TriggIO` kompenzuje zpoždění v kontrolním systému (zpoždění mezi robotem a servem), ale nikoliv zpoždění v externím zařízení. Pro kompenzaci obou zpoždění použijte `TriggEquip`.

Definovaná data se používají pro implementaci do jedné nebo více následných instrukcích `TriggL`, `TriggC`, nebo `TriggJ`

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggIO`:

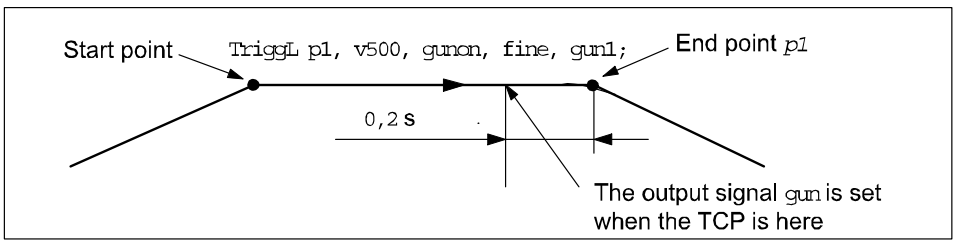
Viz také *Další příklady na str 828*.

Příklad 1

```
VAR triggdata gunon;
...
TriggIO gunon, 0.2\Time\DOp:=gun, 1;
TriggL p1, v500, gunon, fine, gun1;
```

Digitální výstupní signál `gun` je nastaven na hodnotu `1`, když je TCP `0,2` sekund před bodem `p1`.

Obrázek ukazuje příklad I/O události pevné pozice.



xx0500002247

Argumenty

```
TriggIO TriggData Distance [\Start] | [\Time] [\DOp] | [\GOp]|
     [\AOp] | [\ProcID] SetValue | SetDvalue [\DODelay] [\Inhib]
     [\InhibSetValue] [\Mode]
```

`TriggData`

Datový typ: `triggdata`

Proměnná pro uložení `triggdata` vrácených z této instrukce. Tato `triggdata` jsou potom použita v následných instrukcích `TriggL`, `TriggC` nebo `TriggJ`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.295 TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu
RobotWare - OS
Pokračování

`Distance`

Datový typ: `num`

Definuje pozici na dráze, kde by mělo dojít k události I/O.

Určeno jako vzdálenost v mm (kladná hodnota) od koncového bodu dráhy pohybu (použitelné, jestliže není nastaven argument `\Start` nebo `\Time`).

Další podrobnosti najdete v sekcích *Vykonávání programu na str 828* a *Omezení na str 829*.

`[ \Start ]`

Datový typ: `switch`

Používá se, když vzdálenost pro argument `Distance` začíná v bodě začátku pohybu namísto v koncovém bodě.

`[ \Time ]`

Datový typ: `switch`

Používá se, když hodnota určená pro argument `Distance` je ve skutečnosti čas v sekundách (0,5 s) (kladná hodnota) namísto vzdálenosti.

I/O pevné pozice v čase se může použít pouze pro krátké časy (< 0,5 s), předtím, než robot dosáhne koncového bodu instrukce. Více podrobností najdete v sekci Omezení.

`[ \DOp ]`

Digital Output

Datový typ: `signaldo`

Jméno signálu, když digitální výstupní signál bude změněn.

`[ \GOp ]`

Group Output

Datový typ: `signalgo`

Jméno signálu, když skupina digitálních výstupních signál bude změněna.

`[ \AOp ]`

Analog Output

Datový typ: `signalao`

Jméno signálu, když analogový výstupní signál bude změněn.

`[ \ProcID ]`

Process Identity

Datový typ: `num`

Není implementováno pro zákaznické použití.

(Identita IPM procesu pro přijetí události. Selektor je určen v argumentu `SetValue`.)

`SetValue`

Datový typ: `num`

Požadovaná hodnota signálu (v rámci povoleného rozpětí pro aktuální signál). Jestliže signál je digitálním signálem, musí to být hodnota celého čísla. Jestliže

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

signál je digitálním skupinovým signálem, povolená hodnota je závislá na počtu signálů ve skupině. Max hodnota, kterou je možné použít v argumentu `SetValue`, je 8388608, a to je hodnota, kterou může mít 23bitová skupina digitálních signálů jako max hodnotu (viz rozsahy pro `num`).

`SetDvalue`

Datový typ: `dnum`

Požadovaná hodnota signálu (v rámci povoleného rozpětí pro aktuální signál). Jestliže signál je digitálním signálem, musí to být hodnota celého čísla. Jestliže signál je digitálním skupinovým signálem, povolená hodnota je závislá na počtu signálů ve skupině. Max množství signálových bitů, které může skupina digitálních signálů mít, je 32. S proměnnou `dnum` je možné pokrýt rozsah hodnoty 0-4294967295, což je hodnotový rozsah, jaký může 32bitový digitální signál mít.

`[ \DODelay ]`

Digital Output Delay

Datový typ: `num`

Časová prodleva v sekundách (kladná hodnota) pro digitální, skupinový nebo analogový výstupní signál.

Používá se pouze pro prodlevu nastavení výstupních signálů poté, kdy robot dosáhl určené pozice. Jestliže je argument vypuštěn, nedojde k žádné prodlevě.

Prodleva není synchronizována s pohybem.

`[ \Inhib ]`

Inhibit

Datový typ: `bool`

Jméno příznaku perzistentní proměnné pro zpomalení nastavení signálu v čase běhu.

Jestliže je použit tento volitelný argument a aktuální hodnota určeného příznaku je TRUE na pozici-času pro nastavení signálu, potom bude určený signál (`DOp`, `GOp` nebo `AOp`) nastaven na 0 namísto určené hodnoty.

`[ \InhibSetValue ]`

InhibitSetValue

Datový typ: `bool, num or dnum`

Jméno perzistentní proměnné datového typu `bool`, `num` nebo `dnum` nebo kterýkoliv alias těchto tří základnových datových typů.

Tento volitelný argument může být použit pouze společně s volitelným argumentem `Inhib`.

Jestliže je použit tento volitelný argument a hodnota příznaku perzistentní proměnné použité ve volitelném argumentu `Inhib` je TRUE u pozice-čas pro nastavení signálu, hodnota perzistentní proměnné použité ve volitelném argumentu `InhibSetValue` je přečtena a hodnota je použita pro nastavení signálu `DOp`, `GOp` nebo `AOp`.

Jestliže používáme booleánskou perzistentní proměnnou, hodnota TRUE je přeložena na hodnotu 1 a FALSE je přeloženo na hodnotu 0.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.295 TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu
RobotWare - OS
Pokračování

`[ \Mode ]`

Datový typ: `triggmode`

Používá se pro určení odlišných režimů činností při definování spouštěčů.

Vykonávání programu

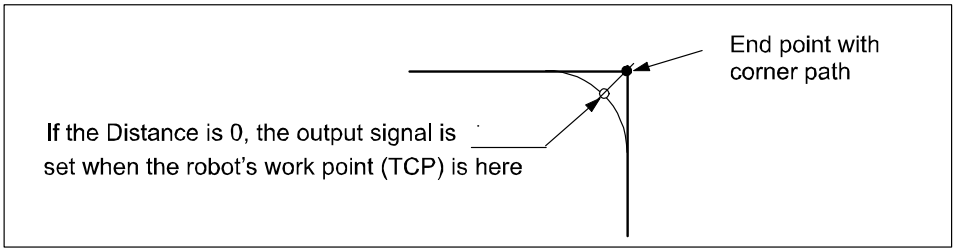
Při provádění instrukce `TriggIO` je podmínka spouštěče uložena do určené proměnné v argumentu `TriggData`.

Později, když se provádí jedna z instrukcí `TriggL`, `TriggC` nebo `TriggJ`, následující jsou použitelné s ohledem na definice v `TriggIO`:

Následující tabulka popisuje vzdálenost určenou v argumentu `Distance`:

Lineární pohyb	Přímá vzdálenosti
Kruhový pohyb	Délka kruhového oblouku
Nelineární pohyb	Přibližná délka oblouku podél dráhy (abychom dostali adekvátní přesnost, vzdálenost by neměla překročit polovinu délky oblouku).

Obrázek ukazuje I/O pevné pozice na rohové dráze.



xx0500002248

I/O pevné pozice bude generováno, když je přejet počáteční bod (koncový bod), jestliže určená vzdálenost od koncového bodu (počátečního bodu) není v rámci délky pohybu aktuální instrukce (`Trigg...`).

Další příklady

Více příkladů jak používat instrukci `TriggIO` je názorně uvedeno dole.

Příklad 1

```
VAR triggdata glueflow;

TriggIO glueflow, 1 \Start \AOp:=glue, 5.3;

MoveJ p1, v1000, z50, tool1;
TriggL p2, v500, glueflow, z50, tool1;
```

Analogový výstupní signál `glue` je nastaven na hodnotu `5.3` , když pracovní bod (TCP) přejede bod umístěný `1` mm za počátečním bodem `p1`.

Příklad 2

```
...
TriggL p3, v500, glueflow, z50, tool1;
```

Analogový výstupní signál `glue` je nastaven ještě jednou na hodnotu `5.3` , když pracovní bod (TCP) přejede bod umístěný `1` mm za počátečním bodem `p2`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Jestliže naprogramovaný argument `SetValue` pro určený analogový výstupní signál `AOp` je mimo limit, potom je systémová proměnná `ERRNO` nastavena na `ERR_AO_LIM`. Tato chyba může být ošetřena v chybovém handleru.

Jestliže naprogramovaný argument `SetValue` nebo `SetDvalue` pro určený digitální skupinový výstupní signál `GOp` je mimo limit, potom je systémová proměnná `ERRNO` nastavena na `ERR_GO_LIM`. Tato chyba může být ošetřena v chybovém handleru.

Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`, potom je systémová proměnná `ERRNO` nastavena na `ERR_NO_ALIASIO_DEF`. Tato chyba může být ošetřena v chybovém handleru.

Omezení

I/O události se vzdáleností (bez argumentu `\Time`) jsou určeny pro průjezdné body (rohová dráha). I/O události se vzdáleností = 0, pomocí stop bodů, opozdí trigg až do doby, kdy robot dosáhne bodu s přesností +/- 24 ms.

I/O události s časem (s argumentem `\Time`) jsou určeny pro stop body. Události přerušení s časem, pomocí průjezdných bodů, mají horší přesnost než je uvedeno dole. I/O události s časem mohou být určeny pouze od koncového bodu pohybu. Tento čas nemůže překročit aktuální brzdný čas robotu, který je max. 0,5 sek. (typické hodnoty při rychlosti 500 mm/s pro IRB2400 150 ms a pro IRB6400 250 ms). Jestliže určený čas je delší než aktuální brzdný čas, potom bude událost přesto generována, ale nikoliv před zahájením brzdění (později než je určeno). Celkový čas pohybu pro aktuální pohyb může být použit během malých a rychlých pohybů.

Typické hodnoty absolutní přesnosti pro sadu digitálních vstupů +/- 5 ms. Typické hodnoty opakované přesnosti pro sadu digitálních vstupů +/- 2 ms.

Syntaxe

```
TriggIO
  [ TriggData ':=' ] < variable (VAR) of triggdata> ','
  [ Distance' :=' ] < expression (IN) of num>
  [ '\' Start ] | [ '\' Time ]
  [ '\' DOp' :=' < variable (VAR) of signaldo> ]
  | [ '\' GOp' :=' < variable (VAR) of signalgo> ]
  | [ '\' AOp' :=' < variable (VAR) of signalao> ]
  | [ '\' ProcID' :=' < expression (IN) of num> ] ','
  [ SetValue' :=' ] < expression (IN) of num>
  | [ SetDvalue' :=' ] < expression (IN) of dnum>
  [ '\' DODelay' :=' < expression (IN) of num> ]
  [ '\' Inhib' :=' < persistent (PERS) of bool> ]
  [ '\' InhibSetValue' :=' < persistent (PERS) of anytype> ]
  [ Mode' :=' ] < expression (IN) of triggmode> ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.295 TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Použití spouštěčů	*TriggL - Lineární pohyby robotu s událostmi na str 839* *TriggC - Kruhový pohyb robotu s událostmi na str 796* *TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Definice I/O události pozice-čas	*TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814*
Definice přerušení se vztahem k pozici	*TriggInt - Definuje přerušení se vztahem k pozici na str 820*
Uložení trigg dat	*triggdata - Polohovací události, trigg na str 1618* *triggmode - Režim činnosti trigg na str 1624*
Definovat kontrolu I/O na pevné pozici	*TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804*
Sada I/O	*SetDO - Mění hodnotu digitálního výstupního signálu na str 629* *SetGO - Mění hodnotu skupiny digitálních výstupních signálů na str 631* *SetAO - Mění hodnotu analogového výstupního signálu na str 620*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.296 TriggJ - Pohyby robotu podle osy s událostmi

Použití

`TriggJ` (*TriggJoint*) se používá pro nastavení výstupních signálů a/nebo provedení rutin přerušení na zhruba pevných pozicích ve stejném čase, kdy se robot rychle pohybuje z jednoho bodu do druhého, když tento pohyb nemusí být v přímé linii.

Jedna nebo více (max. 8) událostí může být definováno pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed`, nebo `TriggRampAO`, a poté je na tyto definice odkázáno v instrukci `TriggJ`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggJ`:

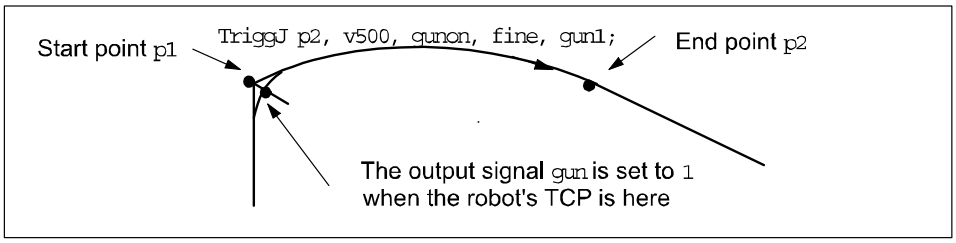
Viz také *Další příklady na str 835*.

Příklad 1

```
VAR triggdata gunon;
...
TriggIO gunon, 0 \Start \DOp:=gun, 1;
MoveL p1, v500, z50, gun1;
TriggJ p2, v500, gunon, fine, gun1;
```

Digitální výstupní signál `gun` je nastaven, když TCP robotu přechází přes midpoint rohové dráhy bodu `p1`.

Obrázek ukazuje příklad I/O události pevné pozice.



xx0500002272

Argumenty

```
TriggJ [\Conc] ToPoint [\ID] Speed [\T] Trigg_1 | TriggArray{*} [
     \T2 ] [ \T3 ] [\T4] [\T5] [\T6] [\T7] [\T8] Zone [\Inpos]
     Tool [\WObj] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datový typ:`switch`

Následné instrukce jsou vykonány za pohybu robotu. Argument je možné použít k zabránění nechtěným zastavením způsobených přetíženým CPU při používání průjezdních bodů. To je výhodné, když naprogramované body jsou při vysokých rychlostech velmi blízko sebe.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.296 TriggJ - Pohyby robotu podle osy s událostmi
RobotWare - OS
Pokračování

Argument je výhodný také když, například, není požadována komunikace s externím zařízením a synchronizace mezi externím zařízením a pohybem robotu. Může se používat také pro ladění vykonávání dráhy robotu, aby se předešlo varování 50024 Závada rohové dráhy nebo chybě 50082 Limit zpomalení.

Použitím argumentu `\Conc` je počet pohybových instrukcí za sebou omezen na 5. V programové sekci, která obsahuje `StorePath-RestoPath`, nejsou pohybové instrukce s argumentem `\Conc` povoleny.

Jestliže je tento argument vypuštěn, potom je následná instrukce vykonána poté, kdy robot dosáhl určeného bodu nebo 100 ms před určenou zónou.

Tento argument se nemůže používat v koordinovaném synchronizovaném pohybu v systému MultiMove.

`ToPoint`

Datový typ:`robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Trigg_1`

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`TriggArray`

Trigg Data Array Parameter

Datový typ: `triggdata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Proměnná pole, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

Omezení je 25 elementů v poli a musí být definováno 1 až 25 spouštěcích podmínek.

Není možné používat volitelné parametry `T2`, `T3`, `T4`, `T5`, `T6`, `T7` nebo `T8` ve stejném čase, kdy je používán argument `TriggArray`.

`[ \T2 ]`

Trigg 2

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T3 ]`

Trigg 3

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T4 ]`

Trigg 4

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T5 ]`

Trigg 5

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T6 ]`

Trigg 6

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T7 ]`

Trigg 7

Datový typ: `triggdata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.296 TriggJ - Pohyby robotu podle osy s událostmi
RobotWare - OS
Pokračování

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`[ \T8 ]`

Trigg 8

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` nebo `TriggRampAO`.

`Zone`

Datový typ:`zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`[ \Inpos ]`

In position

Datový typ: `stoppointdata`

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určené pozice (destinace).

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být stanoven pro pohyb spoje ve vztahu k pracovnímu objektu, který bude proveden.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

V instrukci `MoveJ` najdete více informací o pohybu spoje.

Jelikož spouštěcí podmínky jsou splněny, když je robot umístěn blíž a blíž ke koncovému bodu, definované spouštěcí aktivity jsou provedeny. Spouštěcí podmínky jsou splněny buď v určité vzdálenosti před koncovým bodem instrukce nebo v určité vzdálenosti po bodu startu instrukce nebo v určitém časovém bodu (omezeno na krátký čas) před koncovým bodem instrukce.

Během krokového provádění dopředu jsou I/O aktivity provedeny, ale rutiny přerušení neběží. Během krokového provádění dozadu nejsou prováděny žádné spouštěcí aktivity.

Další příklady

Více příkladů jak používat instrukci `TriggJ` je názorně uvedeno dole.

Příklad 1

```
VAR intnum intno1;
VAR triggdata trigg1;
...
PROC main()
  CONNECT intno1 WITH trap1;
  TriggInt trigg1, 0.1 \Time, intno1;
  ...
  TriggJ p1, v500, trigg1, fine, gun1;
  TriggJ p2, v500, trigg1, fine, gun1;
  ...
  IDelete intno1;
```

Rutina přerušení `trap1` je provedena, když pracovní bod je v pozici 0,1 s před stop bodem `p1` nebo `p2`.

Příklad 2

```
VAR num Distance:=0;
VAR triggdata trigg_array{25};
VAR signaldo myaliassignaldo;
VAR string signalname;
...
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.296 TriggJ - Pohyby robotu podle osy s událostmi

RobotWare - OS

Pokračování

```
PROC main()
  ...
  FOR i FROM 1 TO 25 DO
    signalname:="do";
    signalname:=signalname+ValToStr(i);
    AliasIO signalname, myaliassignaldo;
    TriggEquip trigg_array{i}, Distance \Start, 0
         \DOp:=myaliassignaldo, SetValue:=1;
    Distance:=Distance+10;
  ENDFOR
  TriggJ p1, v500, trigg_array, z30, tool2;
  MoveJ p2, v500, z30, tool2;
  ...
```

Digitální výstupní signály `do1` **až** `do25` **jsou během pohybu nastaveny na** `p1`**. Vzdálenosti mezi nastaveními signálů je 10 mm.**

Řešení chyb

Jestliže naprogramovaný argument `ScaleValue` **pro stanovený analogový výstupní signál** `AOp` **v některé z připojených instrukcí** `TriggSpeed` **je mimo limit pro analogový signál společně s naprogramovaným** `Speed` **v této instrukci, potom je systémová proměnná** `ERRNO` **nastavena na** `ERR_AO_LIM`**.**

Jestliže naprogramovaný argument `DipLag` **v některé z připojených instrukcí** `TriggSpeed` **je příliš velký ve vztahu k Event Present Time použitému v systémových parametrech, potom je systémová proměnná** `ERRNO` **nastavena na** `ERR_DIPLAG_LIM`**.**

Systémová proměnná `ERRNO` **může být nastavena na** `ERR_NORUNUNIT`**, jestliže není kontakt s I/O jednotkou při vstupu do instrukce a použitá triggdata závisí na běžící I/O jednotce, tj. signál se používá v triggdata.**

Jestliže počet pohybových instrukcí za sebou pomocí argumentu `\Conc` **byl překročen, potom je systémová proměnná** `ERRNO` **nastavena na** `ERR_CONC_MAX`**.**

Tyto chyby mohou být zpracovány v chybovém handleru.

Omezení

Jestliže se aktuální bod startu liší od obvyklého, takže celková polohovací délka instrukce `TriggJ` **je kratší než obvykle (například při spuštění** `TriggJ` **s pozicí robotu na koncovém bodě), může se stát, že několik nebo všechny spouštěcí podmínky jsou splněny okamžitě a na stejné pozici. V takových případech bude sekvence, ve které jsou spouštěcí aktivity prováděny, nedefinována. Programová logika v uživatelském programu nesmí být založena na normální sekvenci spouštěcích aktivit pro „nekompletní pohyb“.**

Syntaxe

```
TriggJ
  [ ’\’ Conc ’,’]
  [ ToPoint’ :=’ ] < expression (IN) of robtarget >
  [ ’\’ ID ’:=’ < expression (IN) of identno >]’,’
  [ Speed ’:=’ ] < expression (IN) of speeddata >
  [ ’\’ T ’:=’ < expression (IN) of num > ]’,’
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
[Trigg_1 ':=' ] < variable (VAR) of triggdata > |
[TriggArray ':=' ] < array variable {*} (VAR) of triggdata >
[ '\' T2 ':=' < variable (VAR) of triggdata > ]
[ '\' T3 ':=' < variable (VAR) of triggdata > ]
[ '\' T4 ':=' < variable (VAR) of triggdata > ]
[ '\' T5 ':=' < variable (VAR) of triggdata > ]
[ '\' T6 ':=' < variable (VAR) of triggdata > ]
[ '\' T7 ':=' < variable (VAR) of triggdata > ]
[ '\' T8 ':=' < variable (VAR) of triggdata > ] ´,'
[Zone ':=' ] < expression (IN) of zonedata >
[ '\' Inpos ':=' < expression (IN) of stoppointdata > ]´,'
[ Tool ':=' ] < persistent (PERS) of tooldata >
[ '\' WObj' :='< persistent (PERS) of wobjdata > ]
[ '\' TLoad' :=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Lineární pohyb se spouštěči	*TriggL - Lineární pohyby robotu s událostmi na str 839*
Kruhový pohyb se spouštěči	*TriggC - Kruhový pohyb robotu s událostmi na str 796*
Definice spouštěčů	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814* *TriggRampAO - Definovat událost rampy AO pevné pozice na dráze na str 861* *TriggInt - Definuje přerušení se vztahem k pozici na str 820* *TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804*
Manipulace s `triggdata`	*triggdata - Polohovací události, trigg na str 1618* *TriggDataReset - Resetovat obsah v proměnné trigg-data na str 812* *TriggDataCopy - Kopírovat obsah do proměnné trigg-data na str 810* *TriggDataValid - Zkontrolujte, jestli obsah proměnné triggdata je platný na str 1374*
Posunuje robot pohybem spoje	*MoveJ - Posunuje robot pohybem spoje na str 388*
Pohyb spoje	*Technická referenční příručka - Přehled RAPID*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice dat stop bodu	*stoppointdata - Data stop bodu na str 1590*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovního objektu	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.296 TriggJ - Pohyby robotu podle osy s událostmi

RobotWare - OS

Pokračování

Pro informace o	Viz
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.297 TriggL - Lineární pohyby robotu s událostmi

Použití

`TriggL` (*Trigg Linear*) se používá pro nastavení výstupních signálů a/nebo provádění rutin přerušení na pevných pozicích ve stejném čase, kdy robot provádí lineární pohyb.

Jedna nebo více (max. 8) událostí může být definováno pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO`, nebo `TriggRampAO`. Poté je na tyto definice odkázáno v instrukci `TriggL`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggL`:

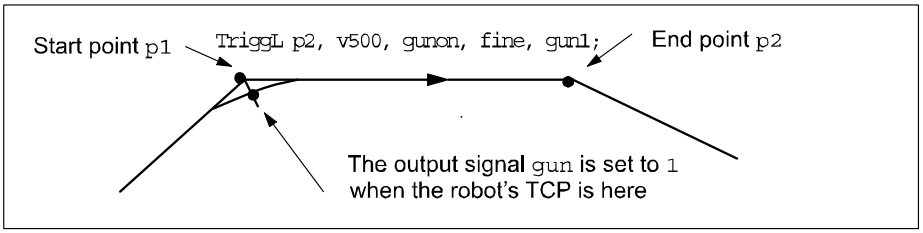
Viz také *Další příklady na str 843*.

Příklad 1

```
VAR triggdata gunon;

TriggIO gunon, 0 \Start \DOp:=gun, 1;
MoveJ p1, v500, z50, gun1;
TriggL p2, v500, gunon, fine, gun1;
```

Digitální výstupní signál `gun` je nastaven, když TCP robotu přechází přes midpoint rohové dráhy bodu `p1`.

Obrázek ukazuje příklad I/O události pevné pozice.



xx0500002291

Argumenty

```
TriggL [\Conc] ToPoint [\ID] Speed [\T] Trigg_1 | TriggArray{*}
     [\T2] [\T3] [\T4] [\T5] [\T6] [\T7] [\T8] Zone [\Inpos] Tool
     [\WObj] [\Corr] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datový typ:`switch`

Následné instrukce jsou vykonány za pohybu robotu. Argument je možné použít k zabránění nechtěným zastavením způsobených přetíženým CPU při používání průjezdních bodů. To je výhodné, když naprogramované body jsou při vysokých rychlostech velmi blízko sebe.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.297 TriggL - Lineární pohyby robotu s událostmi
RobotWare - OS
Pokračování

Argument je výhodný také když, například, není požadována komunikace s externím zařízením a synchronizace mezi externím zařízením a pohybem robotu. Může se používat také pro ladění vykonávání dráhy robotu, aby se předešlo varování 50024 Závada rohové dráhy nebo chybě 50082 Limit zpomalení.

Použitím argumentu `\Conc` je počet pohybových instrukcí za sebou omezen na 5. V programové sekci, která obsahuje `StorePath-RestoPath`, nejsou pohybové instrukce s argumentem `\Conc` povoleny.

Jestliže tento argument je vypuštěn a `ToPoint` není stop bodem, následná instrukce je vykonána předtím, než robot dosáhne naprogramované zóny.

Tento argument se nemůže používat v koordinovaném synchronizovaném pohybu v systému MultiMove.

`ToPoint`

Datový typ:`robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`Trigg_1`

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

`TriggArray`

Trigg Data Array Parameter

Datový typ: `triggdata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Proměnná pole, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

Omezení je 25 elementů v poli a musí být definováno 1 až 25 spouštěcích podmínek.

Není možné používat volitelné parametry `T2`, `T3`, `T4`, `T5`, `T6`, `T7` nebo `T8` ve stejném čase, kdy je používán argument `TriggArray`.

`[ \T2 ]`

Trigg 2

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

`[ \T3 ]`

Trigg 3

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

`[ \T4 ]`

Trigg 4

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

`[ \T5 ]`

Trigg 5

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

`[ \T6 ]`

Trigg 6

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

`[ \T7 ]`

Trigg 7

Datový typ: `triggdata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.297 TriggL - Lineární pohyby robotu s událostmi
RobotWare - OS
Pokračování

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

`[ \T8 ]`

Trigg 8

Datový typ: `triggdata`

Proměnná, která odkazuje ke spouštěcím podmínkám a spouštěcí aktivitě, definované dříve v programu pomocí instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO`.

`Zone`

Datový typ:`zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`[ \Inpos ]`

In position

Datový typ: `stoppointdata`

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určené pozice (destinace).

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být stanoven pro lineární pohyb ve vztahu k pracovnímu objektu, který bude proveden.

`[ \Corr ]`

Correction

Datový typ: `switch`

Korekční data zapsaná do vstupu korekcí instrukcí `CorrWrite` budou přidána k pozici dráhy a destinace, jestliže tento argument je přítomen.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Argument \TLoad popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument \TLoad, potom loaddata v aktuálním tooldata není uvažován.

Když je argument \TLoad nastaven na load0, potom není argument \TLoad uvažován a loaddata v aktuálním tooldata je použit místo toho.

Aby bylo možné použít argument \TLoad, je nezbytné nastavit hodnotu systémového parametru ModalPayLoadMode na 0. Jestliže ModalPayLoadMode je nastaven na 0, není dále možné používat instrukci GripLoad.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr ModalPayLoadMode nastaven na 0, operátor má možnost kopírovat loaddata z nástroje do existující nebo nové proměnné perzistentu loaddata při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému SimMode (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, loaddata ve volitelném argumentu \TLoad není uvažován a loaddata v aktuálním tooldata se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci GripLoad. Proto je výchozí hodnota systémového parametru ModalPayLoadMode 1.

Vykonávání programu

V instrukci MoveL najdete více informací o lineárním pohybu.

Jelikož spouštěcí podmínky jsou splněny, když je robot umístěn blíž a blíž ke koncovému bodu, definované spouštěcí aktivity jsou provedeny. Spouštěcí podmínky jsou splněny buď v určité vzdálenosti před koncovým bodem instrukce nebo v určité vzdálenosti po bodu startu instrukce nebo v určitém časovém bodu (omezeno na krátký čas) před koncovým bodem instrukce.

Během krokového provádění dopředu jsou I/O aktivity provedeny, ale rutiny přerušení neběží. Během krokového provádění dozadu nejsou prováděny žádné spouštěcí aktivity.

Další příklady

Více příkladů jak používat instrukci TriggL je názorně uvedeno dole.

Příklad 1

```
VAR intnum intno1;
VAR triggdata trigg1;
...
PROC main()
  CONNECT intno1 WITH trap1;
  TriggInt trigg1, 0.1 \Time, intno1;
  ...
  TriggL p1, v500, trigg1, fine, gun1;
  TriggL p2, v500, trigg1, fine, gun1;
  ...
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.297 TriggL - Lineární pohyby robotu s událostmi
RobotWare - OS
Pokračování

```
IDelete intno1;
```

Rutina přerušení `trap1` je provedena, když pracovní bod je v pozici `0.1` s před bodem `p1` nebo `p2`.

Příklad 2

```
VAR num Distance:=0;
VAR triggdata trigg_array{25};
VAR signaldo myaliassignaldo;
VAR string signalname;
...
PROC main()
  ...
  FOR i FROM 1 TO 25 DO
    signalname:="do";
    signalname:=signalname+ValToStr(i);
    AliasIO signalname, myaliassignaldo;
    TriggEquip trigg_array{i}, Distance \Start, 0
         \DOp:=myaliassignaldo, SetValue:=1;
    Distance:=Distance+10;
  ENDFOR
  TriggL p1, v500, trigg_array, z30, tool2;
  MoveL p2, v500, z30, tool2;
  ...
```

Digitální výstupní signály `do1` až `do25` jsou během pohybu nastaveny na `p1`. Vzdálenosti mezi nastaveními signálů je 10 mm.

Řešení chyb

Jestliže naprogramovaný argument `ScaleValue` pro stanovený analogový výstupní signál `AOp` v některé z připojených instrukcí `TriggSpeed` je mimo limit pro analogový signál společně s naprogramovaným `Speed` v této instrukci, potom je systémová proměnná `ERRNO` nastavena na `ERR_AO_LIM`.

Jestliže naprogramovaný argument `DipLag` v některé z připojených instrukcí `TriggSpeed` je příliš velký ve vztahu k Event Present Time použitému v systémových parametrech, potom je systémová proměnná `ERRNO` nastavena na `ERR_DIPLAG_LIM`.

Systémová proměnná `ERRNO` může být nastavena na `ERR_NORUNUNIT`, jestliže není kontakt s I/O jednotkou při vstupu do instrukce a použitá triggdata závisí na běžící I/O jednotce, tj. signál se používá v triggdata.

Jestliže počet pohybových instrukcí za sebou pomocí argumentu `\Conc` byl překročen, potom je systémová proměnná `ERRNO` nastavena na `ERR_CONC_MAX`.

Tyto chyby mohou být zpracovány v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Jestliže se aktuální bod startu liší od obvyklého, takže celková polohovací délka instrukce `TriggL` je kratší než obvykle (například při spuštění `TriggL` s pozicí robotu na koncovém bodě), může se stát, že několik nebo všechny spouštěcí podmínky jsou splněny okamžitě a na stejné pozici. V takových případech bude sekvence, ve které jsou spouštěcí aktivity prováděny, nedefinována. Programová logika v uživatelském programu nesmí být založena na normální sekvenci spouštěcích aktivit pro „nekompletní pohyb“.

Syntaxe

```
TriggL
  ['\' Conc ',']
  [ ToPoint' :=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >] ','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ]',’
  [Trigg_1 ':=' ] < variable (VAR) of triggdata > |
  [TriggArray ':=' ] < array variable {*} (VAR) of triggdata >
  [ '\' T2 ':=' < variable (VAR) of triggdata > ]
  [ '\' T3 ':=' < variable (VAR) of triggdata > ]
  [ '\' T4 ':=' < variable (VAR) of triggdata > ]
  [ '\' T5 ':=' < variable (VAR) of triggdata > ]
  [ '\' T6 ':=' < variable (VAR) of triggdata > ]
  [ '\' T7 ':=' < variable (VAR) of triggdata > ]
  [ '\' T8 ':=' < variable (VAR) of triggdata > ] ´,’
  [Zone ':=' ] < expression (IN) of zonedata >
  [ '\' Inpos' :=' < expression (IN) of stoppointdata > ] ´,’
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj' :=' < persistent (PERS) of wobjdata > ]
  [ '\' Corr ]
  [ '\' TLoad' :=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Kruhový pohyb se spouštěči	*TriggC - Kruhový pohyb robotu s událostmi na str 796*
Pohyb spoje se spouštěči	*TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Definice spouštěčů	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814* *TriggInt - Definuje přerušení se vztahem k pozici na str 820* *TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804* *TriggRampAO - Definovat událost rampy AO pevné pozice na dráze na str 861* *TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času na str 868*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.297 TriggL - Lineární pohyby robotu s událostmi

RobotWare - OS

Pokračování

Pro informace o	Viz
Manipulace s `triggdata`	*triggdata - Polohovací události, trigg na str 1618* *TriggDataReset - Resetovat obsah v proměnné triggdata na str 812* *TriggDataCopy - Kopírovat obsah do proměnné triggdata na str 810* *TriggDataValid - Zkontrolujte, jestli obsah proměnné triggdata je platný na str 1374*
Zapisuje do vstupu korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
Lineární pohyb	*Technická referenční příručka - Přehled RAPID*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice dat stop bodu	*stoppointdata - Data stop bodu na str 1590*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Definice dat zóny	*zonedata - Zónová data na str 1642*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.298 TriggJIOs - Společné pohyby robotu s I/O událostmi

Použití

`TriggJIOs` (*Trigg Joint I/O*) se používá pro nastavení výstupních signálů na pevných pozicích ve stejném čase, kdy robot provádí pohyb kloubu.

Instrukce `TriggJIOs` je optimalizována pro poskytnutí dobré přesnosti při používání pohybů se zónami (srovnej s `TriggEquip`/`TriggL`).

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggJIOs`:

Viz také *Další příklady na str 858*.

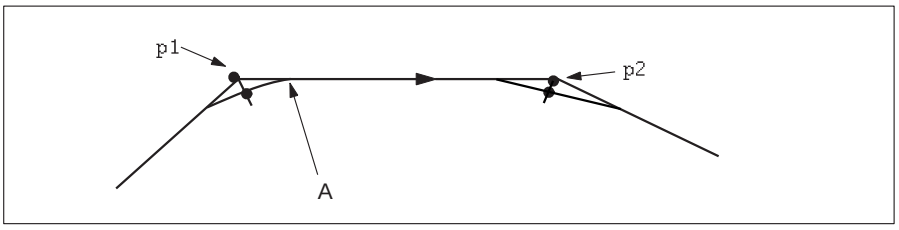
Příklad 1

```
VAR triggios gunon{1};

gunon{1}.used:=TRUE;
gunon{1}.distance:=3;
gunon{1}.start:=TRUE;
gunon{1}.signalname:="gun";
gunon{1}.equiplag:=0;
gunon{1}.setvalue:=1;

MoveJ p1, v500, z50, gun1;
TriggJIOs p2, v500, \TriggData1:=gunon, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Signál `gun` je nastaven, když TCP je 3 mm za bodem `p1`.

Kód RAPID a obrázek ukazují příklad I/O události pevné pozice.

```
TriggJIOs p2, v500, \TriggData1:=gunon, z50, gun1;
```



xx1500000304

A Výstupní signál `gun` je nastaven na `1` když TCP robotu je zde.

Argumenty

```
TriggJIOs ToPoint [\ID] Speed [\T] [\TriggData1] [\TriggData2]
     [\TriggData3] Zone [\Inpos] Tool [\WObj] [\Corr] [\TLoad]
```

`ToPoint`

Datový typ:`robtarget`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.298 TriggJIOs - Společné pohyby robotu s I/O událostmi
RobotWare - OS
Pokračování

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`[\TriggData1]`

Datový typ: `array of triggios`

Proměnná (pole), která odkazuje k podmínkám spouštěče a aktivitě spouštěče. Při používání tohoto argumentu je možné nastavit analogové výstupní signály, digitální výstupní signály a digitální skupinové výstupní signály. Při používání digitálního skupinového výstupního signálu je omezení 23 signálů ve skupině.

`[\TriggData2]`

Datový typ: `array of triggstrgo`

Proměnná (pole), která odkazuje k podmínkám spouštěče a aktivitě spouštěče. Při používání tohoto argumentu je možné nastavit digitální skupinové výstupní signály, které se skládají ze 32 signálů ve skupině a mohou mít max nastavenou hodnotu 4294967295. Mohou se používat pouze digitální skupinové výstupní signály.

`[\TriggData3]`

Datový typ: `array of triggiosdnum`

Proměnná (pole), která odkazuje k podmínkám spouštěče a aktivitě spouštěče. Při používání tohoto argumentu je možné nastavit analogové výstupní signály, digitální výstupní signály a digitální skupinové výstupní signály, které se skládají z 32 signálů ve skupině a mohou mít max nastavenou hodnotu 4294967295.

`Zone`

Datový typ:`zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[ \Inpos ]`

In position

Datový typ: `stoppointdata`

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určené pozice (destinace).

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být stanoven pro lineární pohyb ve vztahu k pracovnímu objektu, který bude proveden.

`[ \Corr ]`

Correction

Datový typ: `switch`

Korekční data zapsaná do vstupu korekcí instrukcí `CorrWrite` budou přidána k pozici dráhy a destinace, jestliže tento argument je přítomen.

`[ \TLoad ]`

Total load

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.298 TriggJIOs - Společné pohyby robotu s I/O událostmi
RobotWare - OS
Pokračování

Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

> **POZNÁMKA**
>
> Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

V instrukci `MoveJ` najdete více informací o pohybu kloubu, *MoveJ - Posunuje robot pohybem spoje na str 388*.

S instrukcí `TriggJIOs` je možné nastavovat 1-50 aktivit spouštěče na I/O signálech podél dráhy od A do B. Signály, které je možné používat, jsou digitální výstupní signály, digitální skupinové výstupní signály a analogové výstupní signály. Podmínky spouštěče jsou splněny buď v určité vzdálenosti před koncovým bodem instrukce nebo v určité vzdálenosti za počátečním bodem instrukce.

Instrukce vyžaduje použití buď argumentu `TriggData1`, `TriggData2` nebo `TriggData3` nebo všech tří. Použití jakéhokoliv trigg je nicméně volitelné. Pro zpomalení použití trigg může být komponent `used` nastaven na FALSE v elementu pole datových typů `triggios`/`triggstrgo`/`triggiosdnum`. Jestliže se nepoužívá žádný element pole, potom se instrukce `TriggJIOs` bude chovat jako `MoveJ` a nebudou provedeny žádné I/O činnosti.

Během krokování programu dopředu jsou provedeny I/O činnosti. Během krokového provádění dozadu nejsou prováděny žádné I/O činnosti.

Při nastavování komponentu `EquipLag` v argumentu `TriggData1`, `TriggData2` nebo `TriggData3` na záporný čas (prodleva), I/O signál může být nastaven za bod destinace (`ToPoint`).

Při používání argumentu `TriggData2` nebo `TriggData3` je možné použít hodnoty až 4294967295, což je max hodnota, kterou může skupina digitálních signálů mít (32 signálů ve skupinovém signálu je max pro systém).

Další příklady

Více příkladů jak používat instrukci `TriggJIOs` je názorně uvedeno dole.

Příklad 1

```
VAR triggios mytriggios{3}:= [[TRUE, 3, TRUE, 0, "go1", 55, 0],
     [TRUE, 15, TRUE, 0, "ao1", 10, 0], [TRUE, 3, FALSE, 0, "do1",
     1, 0]];
...
MoveL p1, v500, z50, gun1;
TriggJIOs p2, v500, \TriggData1:=mytriggios, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Digitální skupinový výstupní signál `go1` bude nastaven na hodnotu 55 3 mm od `p1`. Analogový výstupní signál bude nastaven na hodnotu 10 15 mm od `p1`. Digitální výstupní signál do1 bude nastaven 3 mm od ToPoint `p2`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Příklad 2

```
VAR triggios mytriggios{3}:= [[TRUE, 3, TRUE, 0, "go1", 55, 0],
     [TRUE, 15, TRUE, 0, "ao1", 10, 0], [TRUE, 3, FALSE, 0, "do1",
     1, 0]];
VAR triggstrgo mytriggstrgo{3}:= [[TRUE, 3, TRUE, 0, "go2", "1",
     0], [TRUE, 15, TRUE, 0, "go2", "800000", 0], [TRUE, 4, FALSE,
     0, "go2", "4294967295", 0]];
VAR triggiosdnum mytriggiosdnum{3}:= [[TRUE, 10, TRUE, 0, "go3",
     4294967295, 0], [TRUE, 10, TRUE, 0, "ao2", 5, 0], [TRUE, 10,
     TRUE, 0, "do2", 1, 0]];
...
MoveL p1, v500, z50, gun1;
TriggJIOs p2, v500, \TriggData1:=mytriggios \TriggData2:=
     mytriggstrgo \TriggData3:=mytriggiosdnum, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Digitální skupinový výstupní signál `go1` bude nastaven na hodnotu 55 3 mm od `p1`. Analogový výstupní signál`ao1` bude nastaven na hodnotu 10 15 mm od `p1`. Digitální výstupní signál `do1` bude nastaven 3 mm od `ToPoint p2`. Tyto poziční události jsou nastaveny proměnnou `mytriggios`. Proměnná `mytriggstrgo` nastavuje poziční události, které nastanou 3 a 15 mm od `p1`. Nejprve je signál `go2` nastaven na 1, potom je nastaven na 800000. Signál bude nastaven na hodnotu 4294967295 4 mm od `ToPoint p2`. Toto je max hodnota pro 32bitový digitální výstupní signál. Proměnná `mytriggiosdnum` nastavuje tři poziční události, které nastanou 10 mm od `p1`. Nejprve je signál `go3` nastaven na 4294967295, potom je `ao2` nastaven na 5 a nakonec je `do2` nastaven na 1.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_NORUNUNIT`	jestliže není žádný kontakt s jednotkou I/O.
`ERR_GO_LIM`	jestliže naprogramovaný argument `setvalue` pro určený digitální skupinový výstupní signál `signalname` je mimo limity. (Deklarováno v `TriggData1`, `TriggData2` nebo `TriggData3`)
`ERR_AO_LIM`	jestliže naprogramovaný argument `setvalue` pro určený analogový výstupní signál `signalname` je mimo limity. (Deklarováno v `TriggData1` nebo `TriggData3`)

Omezení

Jestliže se aktuální bod startu liší od obvyklého, takže celková polohovací délka instrukce `TriggJIOs` je kratší než obvykle (například při spuštění `TriggJIOs` s pozicí robotu na koncovém bodě), může se stát, že několik nebo všechny spouštěcí podmínky jsou splněny okamžitě a na stejné pozici. V takových případech bude sekvence, ve které jsou spouštěcí aktivity prováděny, nedefinována. Programová logika v uživatelském programu nesmí být založena na normální sekvenci spouštěcích aktivit pro „nekompletní pohyb“.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.298 TriggJIOs - Společné pohyby robotu s I/O událostmi

RobotWare - OS

Pokračování

Omezení počtu trigg v instrukci `TriggJIOs` je 50 pro každou naprogramovanou instrukci. Jestliže k trigg dochází na bližší vzdálenosti, systém si s tím nemusí poradit. Závisí to na způsobu, jak je pohyb prováděn, na použité rychlosti TCP a jak blízko jsou trigg naprogramovány. Tato omezení existují, ale je obtížné předpovídat, kdy se tyto problémy objeví.

Syntaxe

```
TriggJIOs
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >] ','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ '\' TriggData1 ':=' ] < array {*} (VAR) of triggios >
  [ '\' TriggData2 ':=' ] < array {*} (VAR) of triggstrgo >
  [ '\' TriggData3 ':=' ] < array {*} (VAR) of triggiosdnum >
  [ Zone ':=' ] < expression (IN) of zonedata >
  [ '\' Inpos ':=' < expression (IN) of stoppointdata > ] ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ]
  [ '\' Corr ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Lineární pohyby robotu s I/O událostmi	*TriggLIOs - Lineární pohyby robotu s I/O událostmi na str 854*
Ukládání trigg podmínek a činnosti spouštěče	*triggios - Positioning events, trigg na str 1619*
Ukládání trigg podmínek a činnost spouštěče pro skupinu digitálních signál skládající se z 32 signálů	*triggstrgo - Positioning events, trigg na str 1627*
Ukládání trigg podmínek a činnosti spouštěče	*triggiosdnum - Positioning events, trigg na str 1622*
Alokace objektů událostí	*Technická referenční příručka - Systémové parametry*
Lineární pohyb	*Technická referenční příručka - Přehled RAPID*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.299 TriggLIOs - Lineární pohyby robotu s I/O událostmi

Použití

`TriggLIOs` (Trigg Linear I/O) se používá pro nastavení výstupních signálů na pevných pozicích ve stejném čase, kdy robot provádí lineární pohyb.

Instrukce `TriggLIOs` je optimalizována pro poskytnutí dobré přesnosti při používání pohybů se zónami (srovnej s `TriggEquip`/`TriggL`).

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggLIOs`:

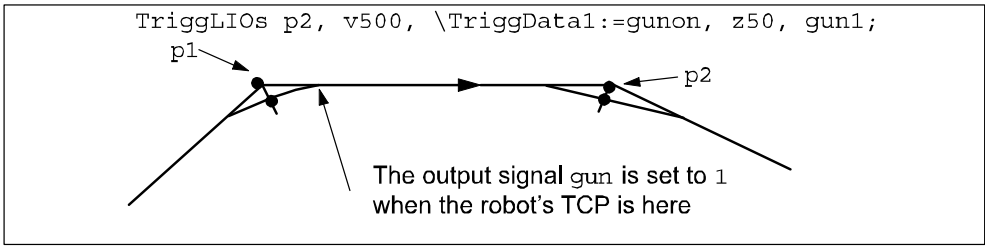
Viz také *Další příklady na str 858*.

Příklad 1

```
VAR triggios gunon{1};

gunon{1}.used:=TRUE;
gunon{1}.distance:=3;
gunon{1}.start:=TRUE;
gunon{1}.signalname:="gun";
gunon{1}.equiplag:=0;
gunon{1}.setvalue:=1;

MoveJ p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData1:=gunon, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Signál `gun` je nastaven, když TCP je 3 mm za bodem `p1` .

Obrázek ukazuje příklad I/O události pevné pozice.



en0800000157

Argumenty

```
TriggLIOs [\Conc] ToPoint [\ID] Speed [\T] [\TriggData1]
     [\TriggData2] [\TriggData3] Zone [\Inpos] Tool [\WObj] [\Corr]
     [\TLoad]
```

`[ \Conc ]`

Concurrent

Datový typ:`switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Následné instrukce jsou vykonány za pohybu robotu. Argument je možné použít k zabránění nechtěným zastavením způsobených přetíženým CPU při používání průjezdních bodů. To je výhodné, když naprogramované body jsou při vysokých rychlostech velmi blízko sebe.

Argument je výhodný také když, například, není požadována komunikace s externím zařízením a synchronizace mezi externím zařízením a pohybem robotu. Může se používat také pro ladění vykonávání dráhy robotu, aby se předešlo varování 50024 Závada rohové dráhy nebo chybě 50082 Limit zpomalení.

Použitím argumentu `\Conc` je počet pohybových instrukcí za sebou omezen na 5. V programové sekci, která obsahuje `StorePath-RestoPath`, nejsou pohybové instrukce s argumentem `\Conc` povoleny.

Jestliže tento argument je vypuštěn a `ToPoint` není stop bodem, následná instrukce je vykonána předtím, než robot dosáhne naprogramované zóny.

Tento argument se nemůže používat v koordinovaném synchronizovaném pohybu v systému MultiMove.

`ToPoint`

Datový typ:`robtarget`

Bod určení robotu a externích os. Je definován jako jmenovitá pozice nebo uložen přímo v instrukci (označeno * v instrukci).

`[ \ID ]`

Synchronization id

Datový typ: `identno`

Argument `[ \ID ]` je povinný v systémech MultiMove, jestliže pohyb je synchronizován nebo koordinovaně synchronizován. Tento argument není povolen v žádném jiném případě. Určené id číslo musí být stejné ve všech úlohách spolupracujících programů. Při používání id čísla nejsou pohyby pomíchány v době běhu.

`Speed`

Datový typ: `speeddata`

Rychlostní data, která se vztahují k pohybům. Rychlostní data definují rychlost TCP, reorientaci nástroje a externí osy.

`[ \T ]`

Time

Datový typ: `num`

Tento argument se používá ke stanovení celkového času v sekundách, během kterého se robot pohybuje. Je potom vyměněn za odpovídající rychlostní data.

`[\TriggData1]`

Datový typ: `array of triggios`

Proměnná (pole), která odkazuje k podmínkám spouštěče a aktivitě spouštěče. Při používání tohoto argumentu je možné nastavit analogové výstupní signály, digitální výstupní signály a digitální skupinové výstupní signály. Při používání digitálního skupinového výstupního signálu je omezení 23 signálů ve skupině.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.299 TriggLIOs - Lineární pohyby robotu s I/O událostmi
RobotWare - OS
Pokračování

`[\TriggData2]`

Datový typ: `array of triggstrgo`

Proměnná (pole), která odkazuje k podmínkám spouštěče a aktivitě spouštěče. Při používání tohoto argumentu je možné nastavit digitální skupinové výstupní signály, které se skládají ze 32 signálů ve skupině a mohou mít max nastavenou hodnotu 4294967295. Mohou se používat pouze digitální skupinové výstupní signály.

`[\TriggData3]`

Datový typ: `array of triggiosdnum`

Proměnná (pole), která odkazuje k podmínkám spouštěče a aktivitě spouštěče. Při používání tohoto argumentu je možné nastavit analogové výstupní signály, digitální výstupní signály a digitální skupinové výstupní signály, které se skládají z 32 signálů ve skupině a mohou mít max nastavenou hodnotu 4294967295.

`Zone`

Datový typ:`zonedata`

Zónová data pro pohyb. Zónová data popisují velikost generované rohové dráhy.

`[ \Inpos ]`

In position

Datový typ: `stoppointdata`

Tento argument se používá pro stanovení konvergenčních kritérií pro pozici TCP robotu ve stop bodu. Data stop bodu nahrazují zónu stanovenou v parametru `Zone`.

`Tool`

Datový typ: `tooldata`

Používaný nástroj při pohybu robotu. Střední bod nástroje je bod, který se pohybuje do určené pozice (destinace).

`[ \WObj ]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci.

Tento argument může být vypuštěn, a jestliže je to tak, pozice je vztažena ke světovému souřadnému systému. Jestliže, na druhé straně, je použit stacionární TCP nebo koordinované externí osy, tento argument musí být stanoven pro lineární pohyb ve vztahu k pracovnímu objektu, který bude proveden.

`[ \Corr ]`

Correction

Datový typ: `switch`

Korekční data zapsaná do vstupu korekcí instrukcí `CorrWrite` budou přidána k pozici dráhy a destinace, jestliže tento argument je přítomen.

`[ \TLoad ]`

Total load

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Datový typ: `loaddata`

Argument `\TLoad` popisuje celkové zatížení použité v pohybu. Celkové zatížení je zatížení nástroje společně s užitečným zatížením, které nástroj nese. Jestliže je použit argument `\TLoad`, potom `loaddata` v aktuálním `tooldata` není uvažován.

Když je argument `\TLoad` nastaven na `load0`, potom není argument `\TLoad` uvažován a `loaddata` v aktuálním `tooldata` je použit místo toho.

Aby bylo možné použít argument `\TLoad`, je nezbytné nastavit hodnotu systémového parametru `ModalPayLoadMode` na 0. Jestliže `ModalPayLoadMode` je nastaven na 0, není dále možné používat instrukci `GripLoad`.

Celkové zatížení může být identifikováno se servisní rutinou LoadIdentify. Jestli je systémový parametr `ModalPayLoadMode` nastaven na 0, operátor má možnost kopírovat `loaddata` z nástroje do existující nebo nové proměnné perzistentu `loaddata` při běhu servisní rutiny.

Je možné nechat program běžet zkušebně bez užiteční zátěže pomocí digitálního vstupního signálu připojeného ke vstupu systému `SimMode` (Simulated Mode). Jestliže je digitální vstupní signál nastaven na 1, `loaddata` ve volitelném argumentu `\TLoad` není uvažován a `loaddata` v aktuálním `tooldata` se použije místo toho.

POZNÁMKA

Výchozí funkčností pro řešení užitečné zátěže je použít instrukci `GripLoad`. Proto je výchozí hodnota systémového parametru `ModalPayLoadMode` 1.

Vykonávání programu

V instrukci `MoveL` najdete více informací o lineárním pohybu.

S instrukcí `TriggLIOs` je možné nastavovat 1-50 aktivit spouštěče na I/O signálech podél dráhy od A do B. Signály, které je možné používat, jsou digitální výstupní signály, digitální skupinové výstupní signály a analogové výstupní signály. Podmínky spouštěče jsou splněny buď v určité vzdálenosti před koncovým bodem instrukce nebo v určité vzdálenosti za počátečním bodem instrukce.

Instrukce vyžaduje použití buď argumentu `TriggData1`, `TriggData2` nebo `TriggData3` nebo všech tří. Použití jakéhokoliv trigg je nicméně volitelné. Pro zpomalení použití trigg může být komponent `used` nastaven na FALSE v elementu pole datových typů `triggios`/`triggstrgo`/`triggiosdnum`. Jestliže se nepoužívá žádný element pole, potom se instrukce `TriggLIOs` bude chovat jako `MoveL` a nebudou provedeny žádné I/O činnosti.

Během krokování programu dopředu jsou provedeny I/O činnosti. Během krokového provádění dozadu nejsou prováděny žádné I/O činnosti.

Při nastavování komponentu `EquipLag` v argumentu `TriggData1`, `TriggData2` nebo `TriggData3` na záporný čas (prodleva), I/O signál může být nastaven za bod destinace (`ToPoint`).

Při používání argumentu `TriggData2` nebo `TriggData3` je možné použít hodnoty až 4294967295, což je max hodnota, kterou může skupina digitálních signálů mít (32 signálů ve skupinovém signálu je max pro systém).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.299 TriggLIOs - Lineární pohyby robotu s I/O událostmi
RobotWare - OS
Pokračování

Další příklady

Více příkladů jak používat instrukci `TriggLIOs` je názorně uvedeno dole.

Příklad 1

```
VAR triggios mytriggios{3}:= [[TRUE, 3, TRUE, 0, "go1", 55, 0],
     [TRUE, 15, TRUE, 0, "ao1", 10, 0], [TRUE, 3, FALSE, 0, "do1",
     1, 0]];
...
MoveL p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData1:=mytriggios, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Digitální skupinový výstupní signál `go1` bude nastaven na hodnotu 55 3 mm od `p1`. Analogový výstupní signál bude nastaven na hodnotu 10 15 mm od `p1`. Digitální výstupní signál do1 bude nastaven 3 mm od ToPoint `p2`.

Příklad 2

```
VAR triggios mytriggios{3}:= [[TRUE, 3, TRUE, 0, "go1", 55, 0],
     [TRUE, 15, TRUE, 0, "ao1", 10, 0], [TRUE, 3, FALSE, 0, "do1",
     1, 0]];
VAR triggstrgo mytriggstrgo{3}:= [[TRUE, 3, TRUE, 0, "go2", "1",
     0], [TRUE, 15, TRUE, 0, "go2", "800000", 0], [TRUE, 4, FALSE,
     0, "go2", "4294967295", 0]];
VAR triggiosdnum mytriggiosdnum{3}:= [[TRUE, 10, TRUE, 0, "go3",
     4294967295, 0], [TRUE, 10, TRUE, 0, "ao2", 5, 0], [TRUE, 10,
     TRUE, 0, "do2", 1, 0]];
...
MoveL p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData1:=mytriggios \TriggData2:=
     mytriggstrgo \TriggData3:=mytriggiosdnum, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Digitální skupinový výstupní signál `go1` bude nastaven na hodnotu 55 3 mm od `p1`. Analogový výstupní signál `ao1` bude nastaven na hodnotu 10 15 mm od `p1`. Digitální výstupní signál `do1` bude nastaven 3 mm od `ToPoint p2`. Tyto poziční události jsou nastaveny proměnnou `mytriggios`. Proměnná `mytriggstrgo` nastavuje poziční události, které nastanou 3 a 15 mm od `p1`. Nejprve je signál `go2` nastaven na 1, potom je nastaven na 800000. Signál bude nastaven na hodnotu 4294967295 4 mm od `ToPoint p2`. Toto je max hodnota pro 32bitový digitální výstupní signál. Proměnná `mytriggiosdnum` nastavuje tři poziční události, které nastanou 10 mm od `p1`. Nejprve je signál `go3` nastaven na 4294967295, potom je `ao2` nastaven na 5 a nakonec je `do2` nastaven na 1.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_NORUNUNIT`	jestliže není žádný kontakt s jednotkou I/O.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Název	Příčina chyby
ERR_GO_LIM	jestliže naprogramovaný argument `setvalue` pro určený digitální skupinový výstupní signál `signalname` je mimo limity. (Deklarováno v `TriggData1`, `TriggData2` nebo `TriggData3`)
ERR_AO_LIM	jestliže naprogramovaný argument `setvalue` pro určený analogový výstupní signál `signalname` je mimo limity. (Deklarováno v `TriggData1` nebo `TriggData3`)
ERR_CONC_MAX	jestliže počet pohybových instrukcí za sebou pomocí argumentu `\Conc` byl překročen.

Omezení

Jestliže se aktuální bod startu liší od obvyklého, takže celková polohovací délka instrukce `TriggLIOs` je kratší než obvykle (například při spuštění `TriggLIOs` s pozicí robotu na koncovém bodě), může se stát, že několik nebo všechny spouštěcí podmínky jsou splněny okamžitě a na stejné pozici. V takových případech bude sekvence, ve které jsou spouštěcí aktivity prováděny, nedefinována. Programová logika v uživatelském programu nesmí být založena na normální sekvenci spouštěcích aktivit pro „nekompletní pohyb“.

Omezení počtu trigg v instrukci `TriggLIOs` je 50 pro každou naprogramovanou instrukci. Jestliže k trigg dochází na bližší vzdálenosti, systém si s tím nemusí poradit. Závisí to na způsobu, jak je pohyb prováděn, na použité rychlosti TCP a jak blízko jsou trigg naprogramovány. Tato omezení existují, ale je obtížné předpovídat, kdy se tyto problémy objeví.

Syntaxe

```
TriggLIOs
  ['\' Conc ',']
  [ ToPoint' :=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >] ','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ]',''
  [ '\' TriggData1' :=' ] < array {*} (VAR) of triggios >
  [ '\' TriggData2' :=' ] < array {*} (VAR) of triggstrgo >
  [ '\' TriggData3' :=' ] < array {*} (VAR) of triggiosdnum >
  [Zone ':=' ] < expression (IN) of zonedata >
  [ '\' Inpos' :=' < expression (IN) of stoppointdata > ] ´,'
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj' :=' < persistent (PERS) of wobjdata > ]
  [ '\' Corr ]
  [ '\' TLoad' :=' < persistent (PERS) of loaddata > ] ';'
```

Související informace

Pro informace o	Viz
Společné pohyby robotu s I/O událostmi	*TriggJIOs - Společné pohyby robotu s I/O událostmi na str 847*
Ukládání trigg podmínek a činnosti spouštěče	*triggios - Positioning events, trigg na str 1619*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.299 TriggLIOs - Lineární pohyby robotu s I/O událostmi

RobotWare - OS

Pokračování

Pro informace o	Viz
Ukládání trigg podmínek a činnost spouštěče pro skupinu digitálních signál skládající se z 32 signálů	*triggstrgo - Positioning events, trigg na str 1627*
Ukládání trigg podmínek a činnosti spouštěče	*triggiosdnum - Positioning events, trigg na str 1622*
Alokace objektů událostí	*Technická referenční příručka - Systémové parametry*, sekce *Téma Pohyb - Plánovač pohybů - Počet interních objektů událostí*
Lineární pohyb	Technická referenční příručka - Přehled RAPID
Pohyb všeobecně	Technická referenční příručka - Přehled RAPID
Definice zátěže	*loaddata - Zátěžová data na str 1523*
Příklad, jak používat `TLoad`, Celková zátěž.	*MoveL - Pohybuje robotem lineárně na str 411*
definování užitečné zátěže pro robot	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
LoadIdentify, servisní rutina pro identifikaci zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*
Vstupní signál systému *SimMode* pro běh robotu v simulovaném režimu bez užitečné zátěže. (Téma I/O, Typ System Input, Akční hodnoty, *SimMode*)	*Technická referenční příručka - Systémové parametry*
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.300 TriggRampAO - Definovat událost rampy AO pevné pozice na dráze

Použití

`TriggRampAO` (*Trigg Ramp Analog Output*) se používá k definování podmínek a činností pro rampování nahoru nebo dolů hodnoty analogového výstupního signálu na pevné pozici podél dráhy pohybu robotu s možností provedení časové kompenzace kvůli zpoždění v externím zařízení.

Definovaná data se používají pro implementaci do jedné nebo více následných instrukcí `TriggL`, `TriggC` nebo `TriggJ`. Kromě těchto instrukcí se může použít také `TriggRampAO` v instrukcích `CapL` nebo `CapC`.

Typ trig činností připojených ke stejné instrukci `TriggL/C/J` může být `TriggRampAO` nebo kterákoliv z instrukcí `TriggIO`, `TriggEquip`, `TriggSpeed`, `TriggInt` nebo `TriggCheckIO`. Je povolen jakýkoliv typ kombinace kromě toho, že je povolena pouze jedna činnost `TriggSpeed` na stejném signálu ve stejné instrukci `TriggL/C/J`.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `TriggRampAO`:

Viz také *Další příklady na str 865*.

Příklad 1

```
VAR triggdata ramp_up;
...
TriggRampAO ramp_up, 0 \Start, 0.1, aolaser1, 8, 15;
MoveL p1, v200, z10, gun1;
TriggL p2, v200, ramp_up, z10, gun1;
```

Analogový signál `aolaser1` zahájí rampování své logické hodnoty nahoru od aktuální hodnoty k nové hodnotě 8, když TCP nástroje `gun1` je 0,1 s od středu rohové dráhy na p1. Celé rampování nahoru bude provedeno, zatímco robot se posune o 15 mm.

Příklad 2

```
VAR triggdata ramp_down;
...
TriggRampAO ramp_down, 15, 0.1, aolaser1, 2, 10;
MoveL p3, v200, z10, gun1;
TriggL p4, v200, ramp_down, z10, gun1;
```

Analogový signál `aolaser1` zahájí rampování své logické hodnoty dolů od aktuální hodnoty k nové hodnotě 2, když TCP nástroje `gun1` je 15 mm od středu rohové dráhy na p4. Celé rampování dolů bude provedeno, zatímco robot se posune o 10 mm.

Argumenty

```
TriggRampAO TriggData Distance [\Start] EquipLag AOutput SetValue
     RampLength [\Time] [\Inhib] [\InhibSetValue] [\Mode]
```

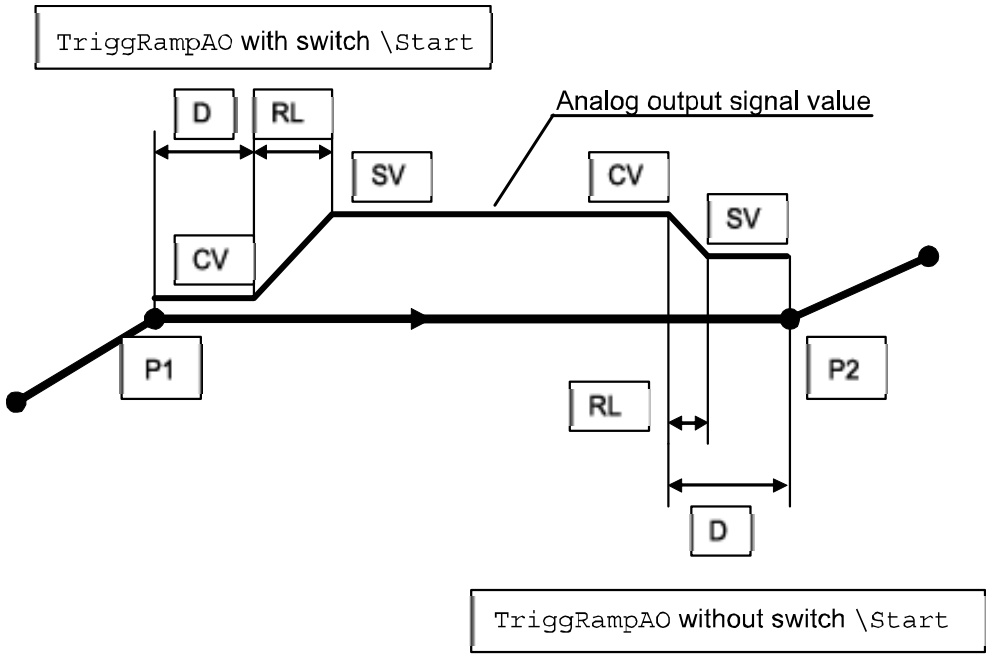
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.300 TriggRampAO - Definovat událost rampy AO pevné pozice na dráze

RobotWare - OS

Pokračování



xx0600003433

D	Parametr `Distance`
RL	Parametr `RampLength`
CV	Aktuální hodnota analogového signálu
SV	Parametr `SetValue` pro hodnotu analogového signálu
P1	`ToPoint` pro předchozí pohybovou instrukci
P2	`ToPoint` pro aktuální instrukci `TrigL/C/J`

`TriggData`

Datový typ: `triggdata`

Proměnná pro uložení `triggdata` vrácených z této instrukce. Tato `triggdata` jsou potom použita v následných instrukcích `TriggL`, `TriggC` , `TriggJ`, `CapL` nebo `CapC`.

`Distance`

Datový typ: `num`

Definuje vzdálenost od středu rohové dráhy, kde bude začínat rampa analogového výstupu.

Určeno jako vzdálenost v mm (kladná hodnota) od koncového bodu (`ToPoint`) dráhy pohybu (použitelné, jestliže není nastaven argument `\Start`).

Další podrobnosti viz *Vykonávání programu na str 864*.

`[\Start]`

Datový typ: `switch`

Používá se, když vzdálenost pro argument `Distance` má vztah k bodu začátku pohybu (předchozí`ToPoint`) namísto ke koncovému bodu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`EquipLag`

Equipment Lag

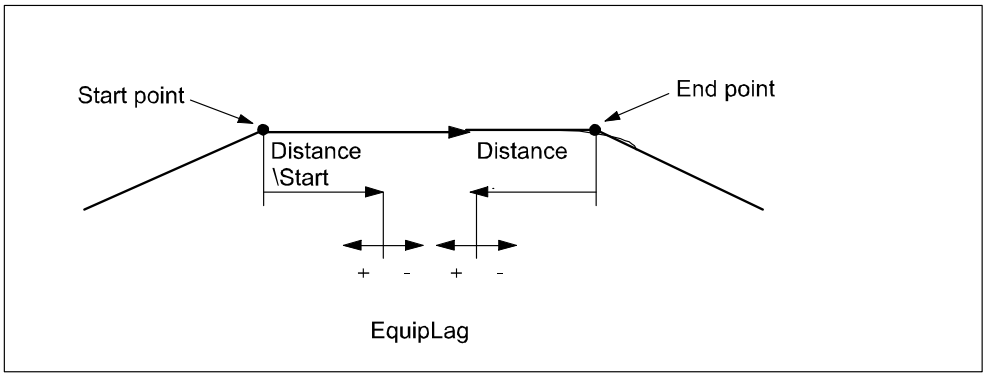
Datový typ: `num`

Určit zpoždění pro externí zařízení v sek.

Pro kompenzaci zpoždění externího zařízení použijte kladnou hodnotu argumentu. Kladná hodnota argumentu znamená, že začátek rampování signálu `AO` je proveden systémem robotu v určeném čase, předtím, než `TCP` fyzicky dosáhne určeného bodu vzdálenosti ve vztahu k počátečnímu nebo koncovému bodu pohybu.

Záporná hodnota argumentu znamená, že začátek rampování signálu `AO` je provedeno systémem robotu v určeném čase. Potom `TCP` fyzicky přešel určený bod vzdálenosti ve vztahu k počátečnímu nebo koncovému bodu pohybu.

Obrázek ukazuje použití argumentu `EquipLag`.



xx0500002262

`AOutput`

Analog Output

Datový typ: `signalao`

Jméno analogového výstupního signálu.

`SetValue`

Datový typ: `num`

Hodnota, na kterou by měl být analogový výstupní signál rampován nahoru nebo dolů (musí být v rámci povolené hodnoty logického rozsahu pro signál). Rampování je spuštěno s aktuální hodnotou analogového výstupního signálu.

`RampLength`

Datový typ: `num`

Délka rampování v mm podél dráhy pohybu `TCP`.

`[\Time]`

Datový typ: `switch`

Jestliže se použije, potom `RampLength` určuje čas rampy v s namísto délky rampování.

Musí se použít, jestliže následný `TriggL, TriggC` nebo `TriggJ` určuje, že celkový pohyb by měl být proveden včas (argument `\T`) namísto rychlosti.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.300 TriggRampAO - Definovat událost rampy AO pevné pozice na dráze
RobotWare - OS
Pokračování

`[ \Inhib ]`

Inhibit

Datový typ: `bool`

Jméno příznaku perzistentní proměnné pro zpomalení nastavení signálu v čase běhu.

Jestliže je použit tento volitelný argument a aktuální hodnota určeného příznaku je TRUE na pozici-času pro začátek rampování I/O signálu, potom bude určený signál (`AOutput`) nastaven na 0.

`[ \InhibSetValue ]`

InhibitSetValue

Datový typ: `bool, num or dnum`

Jméno perzistentní proměnné datového typu `bool`, `num` nebo `dnum` nebo kterýkoliv alias těchto tří základnových datových typů.

Tento volitelný argument může být použit pouze společně s volitelným argumentem `Inhib`.

Jestliže je použit tento volitelný argument a hodnota příznaku perzistentní proměnné ve volitelném argumentu `Inhib` je TRUE u pozice-času pro nastavení signálu, hodnota perzistentní proměnné použité ve volitelném argumentu `InhibSetValue` je přečtena a hodnota je použita pro nastavení signálu `AOutput`.

Jestliže používáme booleánskou perzistentní proměnnou, hodnota TRUE je přeložena na hodnotu 1 a FALSE je přeloženo na hodnotu 0.

`[ \Mode ]`

Datový typ: `triggmode`

Používá se pro určení odlišných režimů činností při definování spouštěčů.

Vykonávání programu

Při provádění instrukce `TriggRampAO` je podmínka spouštěče uložena do určené proměnné pro argument `TriggData`.

Později, když se provádí jedna z instrukcí `TriggL`, `TriggC` nebo `TriggJ`, následující jsou použitelné s ohledem na definice v `TriggRampAO`:

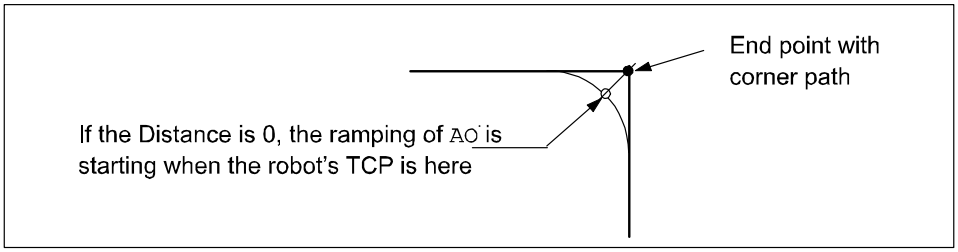
Tabulka popisuje vzdálenost určenou v argumentu `Distance`:

Lineární pohyb	Přímá vzdálenosti
Kruhový pohyb	Délka kruhového oblouku
Nelineární pohyb	Přibližná délka oblouku podél dráhy (abychom dostali adekvátní přesnost, vzdálenost by neměla překročit polovinu délky oblouku).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Obrázek ukazuje rampování AO v rohové dráze.



xx0600003439

Vlastnosti vykonávání programu TriggRampAO připojeného ke kterémukoliv TriggL/C/J:

- Rampování AO je spuštěno, když robot dosáhne určeného bodu Distance na dráze robotu (s kompenzací určeného EquipLag)
- Rampovací funkce bude provedena během časového období vypočítaného z určené RampLength a naprogramované rychlosti TCP. Výpočet bere v úvahu VelSet, ruční potlačení rychlosti a max. 250 mm/s v režimu MAN, ale žádná jiná omezení rychlosti.
- Aktualizace hodnoty signálu AO od počáteční (právě přečtené) hodnoty k určené SetValue bude provedena každých 10 ms s výsledkem ve schodovité formě. Jestliže vypočítaný čas rampy nebo určený čas rampy je delší než 0,5 s, potom bude frekvence rampování zpomalena:
  - <= 0,5 s dává max. 50 kroků každých 10 ms
  - <= 1 s dává max. 50 kroků každých 20 ms
  - <= 1,5 s dává max. 50 kroků každých 30 ms a tak dále

Činnost TriggRampAO je také provedena v kroku FWD, ale nikoliv v krokovém režimu BWD.

V jakémkoliv typu zastavení (ProgStop, Nouzové zastavení ...), jestliže rampovací funkce je aktivní kvůli:

- při rampování nahoru je AO nastaveno krátce na starou hodnotu.
- při rampování dolů je AO nastaveno krátce na novou hodnotu SetValue.

Další příklady

Více příkladů jak používat instrukci TriggRampAO je názorně uvedeno dole.

Příklad 1

```
VAR triggdata ramp_up;
VAR triggdata ramp_down;
...
TriggRampAO ramp_up, 0 \Start, 0.1, aolaser1, 8, 15;
TriggRampAO ramp_down, 15, 0.1, aolaser1, 2, 10;
MoveL p1, v200, z10, gun1;
TriggL p2, v200, ramp_up, \T2:=ramp_down, z10, gun1;
```

V tomto příkladu je rampování nahoru a rampování dolů AO prováděno ve stejné instrukci TriggL na stejné dráze pohybu. Funguje to bez jakékoliv interference nastavení AO, jestliže dráha pohybu je dostatečně dlouhá.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.300 TriggRampAO - Definovat událost rampy AO pevné pozice na dráze
RobotWare - OS
Pokračování

Analogový signál aolaser1 zahájí rampování své logické hodnoty nahoru od aktuální hodnoty k nové hodnotě 8, když TCP nástroje gun1 je 0,1 s před středem rohové dráhy na p1. Celé rampování nahoru bude provedeno, zatímco robot se posune o 15 mm.

Analogový signál aolaser1 zahájí rampování své logické hodnoty dolů od aktuální hodnoty 8 k nové hodnotě 2, když TCP nástroje gun1 je 15 mm plus 0,1 s před středem rohové dráhy na p2. Celé rampování nahoru bude provedeno, zatímco robot se posune o 10 mm.

Řešení chyb

Jestliže naprogramovaný argument SetValue pro určený analogový výstupní signál AOutput je mimo limit, potom je systémová proměnná ERRNO nastavena na ERR_AO_LIM. Tato chyba může být ošetřena v chybovém handleru.

Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO, potom je systémová proměnná ERRNO nastavena na ERR_NO_ALIASIO_DEF. Tato chyba může být ošetřena v chybovém handleru.

Omezení

Hodnota analogového výstupního signálu nebude kompenzována na nižší rychlost TCP v rohové dráze nebo během jiných fází zrychlení nebo zpomalení (AO není úměrný k TCP).

Pouze počáteční bod rampování AO bude proveden na určené pozici na dráze. Rampování nahoru nebo dolů bude provedeno s „mrtvou kalkulací“ s vysokou přesností:

- Při konstantní rychlosti bude odchylka pro konec rampování AO malá ve srovnání s určenou.
- Během fází zrychlování nebo zpomalování, jako jsou body blízko zastavení, bude odchylka větší.
- Doporučení: před rampováním nahoru a po rampování dolů použijte rohové dráhy.

Jestliže používáme dva nebo několik TriggRampAO na stejném analogovém výstupním signálu a jsme připojeni ke stejné instrukci TriggL/C/J, a oba nebo několik RampLength je umístěno na stejné části dráhy robotu, potom se budou nastavení AO vzájemně ovlivňovat,

Událost rampy se vztahem k pozici (+/- čas) AO bude spuštěna, když je předchozí ToPoint přejet, jestliže určená Distance od aktuálního ToPoint není v rámci délky pohybu pro aktuální instrukci TriggL/C/J . Událost rampy se vztahem k pozici (+/- čas) AO bude spuštěna, když aktuální ToPoint je přejet, jestliže určená Distance od předchozího ToPoint není v rámci délky pohybu pro aktuální instrukci TriggL/C/J (s argumentem \Start).

Není podpora pro restart rampovací funkce AO po jakémkoliv typu zastavení (ProgStop, Nouzové zastavení ...).

Při restartu po výpadku napájení je instrukce TriggL/C/J spuštěna od začátku aktuální pozice výpadku napájení.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
TriggRampAO
  [ TriggData ':=' ] < variable (VAR) of triggdata > ','
  [ Distance ':=' ] < expression (IN) of num >
  [ '\' Start ] ','
  [ EquipLag ':=' ] < expression (IN) of num > ','
  [ AOutput ':=' ] < variable (VAR) of signalao> ','
  [ SetValue ':=' ] < expression (IN) of num> ','
  [ RampLength ':=' ] < expression (IN) of num> ','
  [ '\' Time ]
  [ '\' Inhib' :=' < persistent (PERS) of bool> ]
  [ '\' InhibSetValue' :=' < persistent (PERS) of anytype> ]
  [ Mode' :=' ] < expression (IN) of triggmode> ';'
```

Související informace

Pro informace o	Viz
Použití spouštěčů	*TriggL - Lineární pohyby robotu s událostmi na str 839* *TriggC - Kruhový pohyb robotu s událostmi na str 796* *TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Definice ostatních trigg	*TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814*
Uložení triggdata	*triggdata - Polohovací události, trigg na str 1618* *triggmode - Režim činnosti trigg na str 1624*
Sada analogových výstupních signálů	*SetAO - Mění hodnotu analogového výstupního signálu na str 620* *signalxx - Digitální a analogové signály na str 1582*
Konfigurace přednastaveného času události	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.301 TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času

Použití

`TriggSpeed` se používá k definování podmínek a činností pro kontrolu analogového výstupního signálu s výstupní hodnotou, která je proporční s aktuální rychlostí TCP. Začátek, škálování a zakončení analogového výstupu může být určeno na pevné pozici-času podél dráhy pohybu robotu. Je možné použít časovou kompenzaci pro zpoždění v externím zařízení pro začátek, škálování a zakončení analogového výstupu a také pro rychlostní propady robotu.

Definovaná data se používají v jedné nebo více následných instrukcích `TriggL`, `TriggC`, nebo `TriggJ` .

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Základní příklady

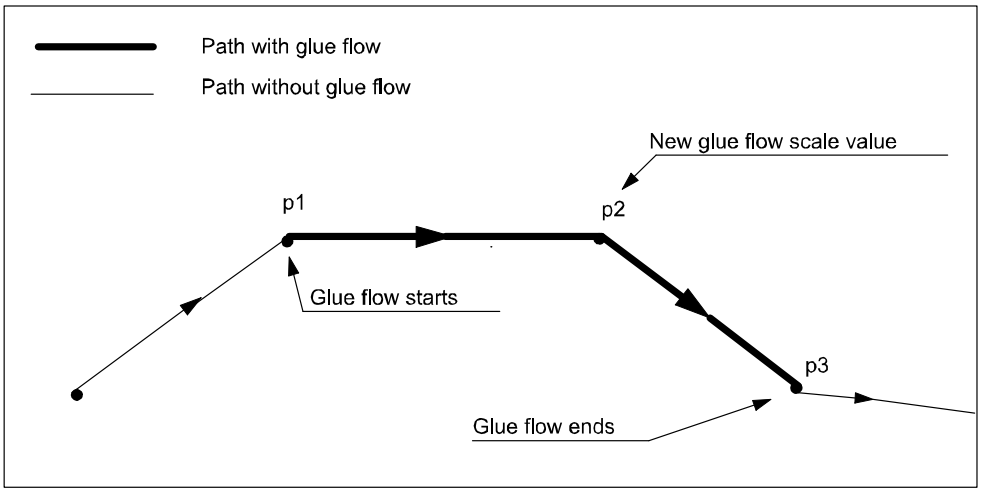
Následující příklad názorně ukazuje instrukci `TriggSpeed`:

Viz také *Další příklady na str 873*.

Příklad 1

```
VAR triggdata glueflow;
TriggSpeed glueflow, 0, 0.05, glue_ao, 0.8\DipLag:=0.04
     \ErrDO:=glue_err;
TriggL p1, v500, glueflow, z50, gun1;
TriggSpeed glueflow, 10, 0.05, glue_ao, 1;
TriggL p2, v500, glueflow, z10, gun1;
TriggSpeed glueflow, 0, 0.05, glue_ao, 0;
TriggL p3, v500, glueflow, z50, gun1;
```

Obrázek dole ilustruje příklad sekvence `TriggSpeed`



xx0500002329

Průtok lepidla (analogový výstup `glue_ao`) s hodnotou škály `0.8` začíná, když TCP je `0.05` s před bodem `p1`, nová škálová hodnota průtoku lepidla `1`, když TCP

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

je `10` mm plus `0.05` s před bodem `p2`, a průtok lepidla končí (škálová hodnota 0), když TCP je `0.05` s před bodem `p3`.

Každý propad rychlosti robotu je časově kompenzován takovým způsobem, aby analogový výstupní signál `glue_ao` byl ovlivněn `0.04` s před tím, než se objeví propad rychlosti TCP.

Jestliže dojde k přeplnění vypočítané hodnoty logického analogového výstupu v `glue_ao`, potom je nastaven digitální výstupní signál `glue_err`. Jestliže už se nevyskytuje žádné přeplnění, potom je znovu nastaven `glue_err`.

Argumenty

```
TriggSpeed TriggData Distance [\Start] ScaleLag AOp ScaleValue
    [\DipLag] [\ErrDO] [\Inhib] [\InhibSetValue] [\Mode]
```

`TriggData`

Datový typ: `triggdata`

Proměnná pro uložení `triggdata` vrácených z této instrukce. Tato `triggdata` jsou potom použita v následných instrukcích `TriggL`, `TriggC` nebo `TriggJ`.

`Distance`

Datový typ: `num`

Definuje pozici dráhy pro změnu analogové výstupní hodnoty.

Určeno jako vzdálenost v mm (kladná hodnota) od koncového bodu dráhy pohybu (použitelné, jestliže není nastaven argument `Start`).

Další podrobnosti viz *Vykonávání programu na str 871*.

`[ \Start ]`

Datový typ: `switch`

Používá se, když vzdálenost pro argument `Distance` začíná v bodě začátku pohybu namísto v koncovém bodě.

`ScaleLag`

Datový typ: `num`

Určete zpoždění jako čas v s (kladná hodnota) v externím zařízení pro změnu hodnoty analogového výstupu (spuštění, škálování a ukončení).

Pro kompenzaci zpoždění externího zařízení znamená tato hodnota argumentu, že analogový výstupní signál je nastaven robotem na určený čas, předtím, než TCP fyzicky dosáhne určené vzdálenosti ve vztahu k počátečnímu nebo koncovému bodu pohybu.

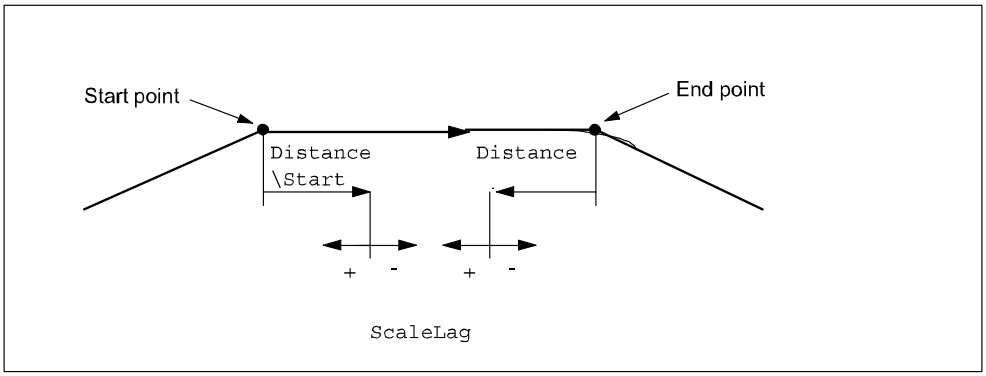
Argument je možné používat také k rozšíření analogového výstupu za koncový bod. Nastavte čas v sekundách, po který by robot měl zachovat analogový výstup. Čas nastavte se záporným znaménkem. Limit je -0,10 sekundy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.301 TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času
RobotWare - OS
Pokračování

Obrázek dole ukazuje použití argumentu `ScaleLag`



xx0500002330

`AOp`

Analog Output

Datový typ: `signalao`

Jméno analogového výstupního signálu.

`ScaleValue`

Datový typ: `num`

Škálovací hodnota pro analogový výstupní signál.

Hodnota fyzického výstupu pro analogový signál je vypočítána robotem:

- Hodnota logického výstupu = Škálovací hodnota * Aktuální rychlost TCP v mm/s.
- Hodnota fyzického výstupu = Podle definice v konfiguraci pro aktuální analogový výstupní signál se shora uvedenou hodnotou logického výstupu jako vstupem.

`[ \DipLag ]`

Datový typ: `num`

Určete zpoždění jako čas v s (kladná hodnota) pro externí zařízení při změně hodnoty analogového výstupu, protože rychlost robotu má propady.

Pro kompenzaci zpoždění externího zařízení znamená tato hodnota argumentu, že analogový výstupní signál je nastaven robotem v určeném čase, předtím, než se objeví propad rychlosti TCP.

Tento argument může používat pouze robot pro první `TriggSpeed` (v kombinaci s jedním z `TriggL`, `TriggC`, or `TriggJ`) v sekvenci několika instrukcí `TriggSpeed`. Hodnota prvního určeného argumentu je platná pro všechny následující `TriggSpeed` v sekvenci.

`[ \ErrDO ]`

Error Digital Output

Datový typ: `signaldo`

Jméno digitálního výstupního signálu pro hlášení přeplnění analogové hodnoty.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže během pohybu je výsledkem výpočtu hodnoty logického analogového výstupu pro signál v argumentu `AOp` přeplnění kvůli nadměrné rychlosti, potom je tento signál nastaven a hodnota fyzického analogového výstupu je snížena na max hodnotu. Jestliže dále už nedochází k přeplňování, signál je znovu nastaven.

Tento argument může používat pouze robot pro první `TriggSpeed` (v kombinaci s jedním z `TriggL`, `TriggC`, or `TriggJ`) v sekvenci několika instrukcí `TriggSpeed`. Hodnota prvního daného argumentu je platná pro všechny následující `TriggSpeed` v sekvenci.

`[ \Inhib ]`

Inhibit

Datový typ: `bool`

Jméno příznaku perzistentní proměnné pro zpomalení nastavení analogového signálu v čase běhu.

Jestliže je použit tento volitelný argument a aktuální hodnota určeného příznaku je `TRUE` v době pro nastavení analogového signálu, potom bude určený signál `AOp`) nastaven na 0 namísto vypočítané hodnoty.

Tento argument může používat pouze robot pro první `TriggSpeed` (v kombinaci s jedním z `TriggL`, `TriggC`, or `TriggJ`) v sekvenci několika instrukcí `TriggSpeed`. Hodnota prvního daného argumentu je platná pro všechny následující `TriggSpeed` v sekvenci.

`[ \InhibSetValue ]`

InhibitSetValue

Datový typ: `bool, num or dnum`

Jméno perzistentní proměnné datového typu `bool`, `num` nebo `dnum` nebo kterýkoliv alias těchto tří základnových datových typů.

Tento volitelný argument může být použit pouze společně s volitelným argumentem `Inhib`.

Jestliže je použit tento volitelný argument a hodnota příznaku perzistentní proměnné ve volitelném argumentu `Inhib` je TRUE u pozice-času pro nastavení signálu, hodnota perzistentní proměnné použité ve volitelném argumentu `InhibSetValue` je přečtena a hodnota je použita pro nastavení signálu `AOp`.

Jestliže používáme booleánskou perzistentní proměnnou, hodnota TRUE je přeložena na hodnotu 1 a FALSE je přeloženo na hodnotu 0.

`[ \Mode ]`

Datový typ: `triggmode`

Používá se pro určení odlišných režimů činností při definování spouštěčů.

Vykonávání programu

Při provádění instrukce `TriggSpeed` je podmínka spouštěče uložena do určené proměnné pro argument `TriggData`.

Později, když se provádí jedna z instrukcí `TriggL`, `TriggC` nebo `TriggJ`, následující jsou použitelné s ohledem na definice v `TriggSpeed`:

Pokračování na další straně

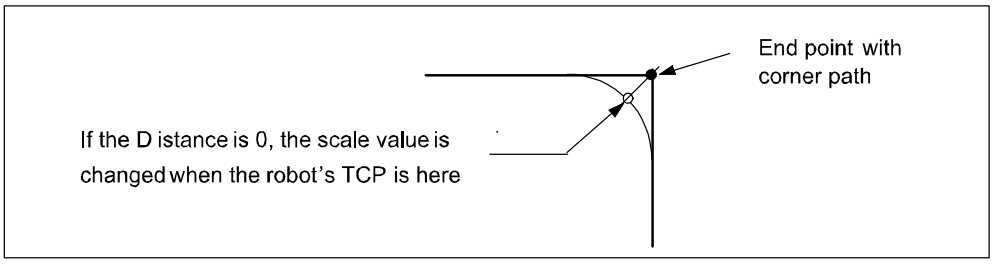
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.301 TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času
RobotWare - OS
Pokračování

Pokud jde o vzdálenosti určenou v argumentu `Distance`, viz tabulku dole:

Lineární pohyb	Přímá vzdálenosti
Kruhový pohyb	Délka kruhového oblouku
Nelineární pohyb	Přibližná délka oblouku podél dráhy (abychom dostali adekvátní přesnost, vzdálenost by neměla překročit polovinu délky oblouku).

Obrázek dole ilustruje hodnotovou událost škály pevné pozice-času na rohové dráze.

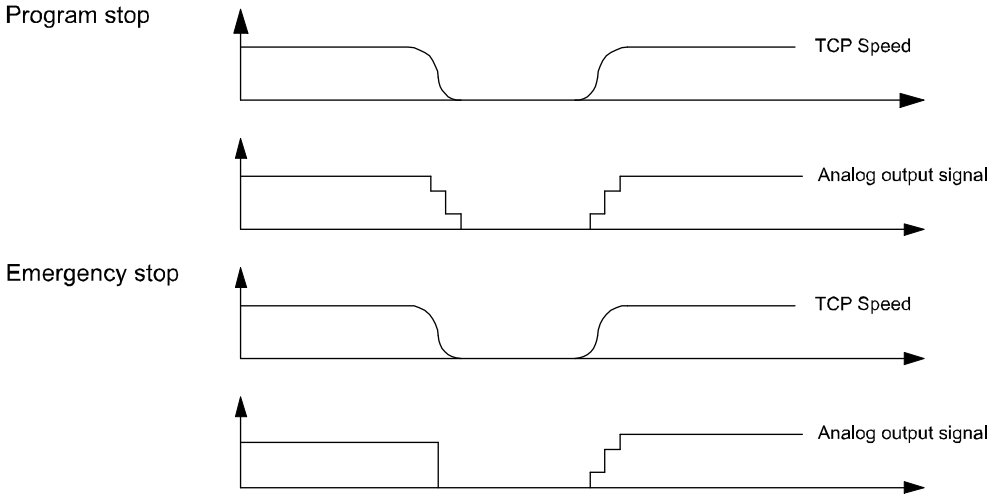


xx0500002331

Hodnotová událost škály ve vztahu k pozici-času bude generována, když je přejet počáteční bod (koncový bod), jestliže určená vzdálenost od koncového bodu (počátečního bodu) není v rámci délky pohybu aktuální instrukce (`TriggL`, `TriggC`, nebo `TriggJ`).

První `TriggSpeed` použitá jednou z instrukcí `TriggL`, `TriggC` nebo `TriggJ` bude interně v systému vytvářet proces se stejným jménem jako má analogový výstupní signál. Stejný proces bude použit všemi následnými `TriggL`, `TriggC` nebo `TriggJ`, což odkazuje ke stejnému jménu signálu a nastavení od instrukce `TriggSpeed` .

Proces okamžitě nastaví analogový výstup na 0, v případě nouzového zastavení programu. V případě zastavení programu zůstane analogový výstupní signál proporční s rychlostí TCP, dokud robot nebude stát v klidu. Proces zůstane „živý“ a bude připraven na restart. Když se robot restartuje, signál je proporční s rychlostí TCP přímo od začátku.



xx0500002332

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Proces „zemře“ po zpracování události škály s hodnotou 0, jestliže žádná následujících `TriggL`, `TriggC` nebo `TriggJ` není v té době ve frontě.

Další příklady

Více příkladů instrukce `TriggSpeed` je názorně uvedeno dole.

Příklad 1

```
VAR triggdata flow;
TriggSpeed flow, 10 \Start, 0.05, flowsignal, 0.5 \DipLag:=0.03;
MoveJ p1, v1000, z50, tool1;
TriggL p2, v500, flow, z50, tool1;
```

Analogový výstupní signál `flowsignal` je nastaven na logickou hodnotu = (`0.5` * aktuální rychlost TCP v mm/s) `0.05` s před projetím TCP bodem umístěným `10` mm za počátečním bodem `p`. Výstupní hodnota je upravena, aby byla proporční k aktuální rychlosti TCP během pohybu k `p2`.

```
...
TriggL p3, v500, flow, z10, tool1;
```

Robot se pohybuje od `p2` k `p3` s hodnotou analogového výstupu proporční k aktuální rychlosti TCP. Hodnota analogového výstupu bude snížena v čase `0.03` s předtím, než robot sníží rychlost TCP během procházení rohovou drahou `z10`.

Omezení

Omezení pro instrukci `TriggSpeed` jsou uvedena dole.

Přesnost události škálové hodnoty vztažené k pozici-času

Typické hodnoty absolutní přesnosti pro události škálové hodnoty ±5 ms.

Typické hodnoty absolutní přesnosti pro události škálové hodnoty ±2 ms.

Přesnost adaptace propadů rychlosti TCP (fáze zpomalení - zrychlení)

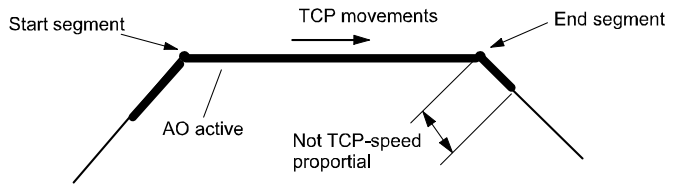
Typické hodnoty absolutní přesnosti pro adaptaci propadů rychlosti TCP ±5 ms.

Typické hodnoty opakované přesnosti pro adaptaci propadů rychlosti TCP ±2ms (hodnota závisí na konfigurovaném *Path resolution*).

Záporný `ScaleLag`

Jestliže negativní hodnota na parametru `ScaleLag` je použita pro přesun nulového škálování k dalšímu příkazu pohybu, potom nebude analogový výstupní signál resetován, jestliže se objeví zastavení programu. Nouzové zastavení vždy resetuje analogový signál.

Analogový signál není nadále proporcionální s rychlostí TCP po koncovém bodu na příkazu pohybu.



xx0500002333

Pokračování na další straně

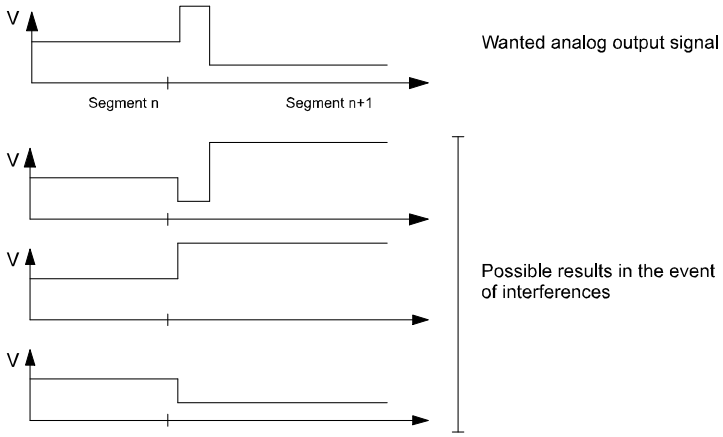
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.301 TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času
RobotWare - OS
Pokračování

Řešení chyb

Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`, potom je systémová proměnná `ERRNO` nastavena na `ERR_NO_ALIASIO_DEF`. Tato chyba může být ošetřena v chybovém handleru.

Jsou dány dva po sobě jdoucí pohybové příkazy s instrukcemi `TriggL`/`TriggSpeed`. Záporná hodnota v parametru `ScaleLag` umožňuje posunout škálovou událost od prvního pohybového příkazu na začátek druhého pohybového příkazu. Jestliže druhý pohybový příkaz škáluje na začátku, potom není kontrola, jestli se dvě škály ruší.



xx0500002334

Související systémové parametry

Systémový parametr *Event Preset Time* se používá pro zpoždění robotu, aby bylo umožněno aktivovat/kontrolovat externí vybavení předtím, než robot projede pozicí.

Tabulka dole ukazuje doporučení pro nastavení systémového parametru *Event Preset Time*, kde typické zpoždění serva je 0,040 s.

`ScaleLag`	`DipLag`	Požadovaný *Event Preset Time* pro vyloučení chyby vykonávání za běhu	Doporučený *Event Preset Time* pro získání nejlepší přesnosti
`ScaleLag > DipLag`	Vždy	`DipLag`, jestliže`DipLag > ServoLag`	`ScaleLag` v s plus 0,090 s
`ScaleLag < DipLag`	`DipLag < Servo Lag`	`- " -`	0,090 s
`- " -`	`DipLag > Servo Lag`	`- " -`	`DipLag` v s plus 0,030 s

Syntaxe

```
TriggSpeed
  [ TriggData ':=' ] < variable (VAR) of triggdata>','
  [ Distance' :=' ] < expression (IN) of num>
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.301 TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času


```
[ '\' Start ] ','
[ ScaleLag':=' ] < expression (IN) of num> ','
[ AOp ':='] < variable (VAR) of signalao> ','
[ ScaleValue' :=' ] < expression (IN) of num>
[ '\' DipLag' :=' < expression (IN) of num> ]
[ '\' ErrDO' :=' < variable (VAR ) of signaldo> ]
[ '\' Inhib' :=' < persistent (PERS ) of bool >]
[ '\' InhibSetValue' :=' < persistent (PERS) of anytype> ]
[ Mode' :=' ] < expression (IN) of triggmode> ';'
```

Související informace

Pro informace o	Viz
Použití spouštěčů	*TriggL - Lineární pohyby robotu s událostmi na str 839* *TriggC - Kruhový pohyb robotu s událostmi na str 796* *TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Definice ostatních trigg	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggInt - Definuje přerušení se vztahem k pozici na str 820* *TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814*
Uložení triggdata	*triggdata - Polohovací události, trigg na str 1618* *triggmode - Režim činnosti trigg na str 1624*
Konfigurace Event preset time	*Technická referenční příručka - Systémové parametry*
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.302 TriggStopProc - Generovat restartovací data pro trigg signály při zastavení

Použití

Instrukce `TriggStopProc` vytváří interní proces supervize v systému pro nulové nastavování určitých procesních signálů a vygenerování dat restartu v určené perzistentní proměnné při každém zastavení programu (`STOP`) nebo nouzovém zastavení (`QSTOP`) v systému.

`TriggStopProc` a datový typ `restartdata` jsou určeny pro použití k restartu po zastavení programu (`STOP`) nebo nouzovém zastavení (`QSTOP`) vlastních procesních instrukcí definovaných v RAPIDu (rutiny `NOSTEPIN`).

Je možné v uživatelsky definované událostní rutině `RESTART` analyzovat aktuální restartovací data, krok dozadu na dráze s instrukcí `StepBwdPath` a aktivovat vhodné procesní signály před restartem pohybu.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* ve všech pohybových úlohách.

Všimněte si u systému MultiMove, že pouze jeden podpůrný proces `TriggStopProc` s určeným jménem stínového signálu (argument `ShadowDO`) může být aktivní v systému ve stejnou dobu. To znamená, že `TriggStopProc` dohlíží na zastavení programu nebo nouzové zastavení v programové úloze, kde byl naposledy vykonáván.

Argumenty

```
TriggStopProc RestartRef [\DO] [\GO1] [\GO2] [\GO3] [\GO4] ShadowDO
```

`RestartRef`

Restart Reference

Datový typ: `restartdata`

Perzistentní proměnná, ve které budou data restartu přístupná po každém zastavení vykonávání programu.

`[\DO1]`

Digital Output 1

Datový typ: `signaldo`

Proměnná signálu pro digitální procesní signál, který bude nastaven na nulu a dohlížen v datech restartu, když vykonávání programu je zastaveno.

`[\GO1]`

Group Output 1

Datový typ: `signalgo`

Proměnná signálu pro digitální skupinový procesní signál, který bude nastaven na nulu a dohlížen v datech restartu, když vykonávání programu je zastaveno.

`[\GO2]`

Group Output 2

Datový typ: `signalgo`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Proměnná signálu pro digitální skupinový procesní signál, který bude nastaven na nulu a dohlížen v datech restartu, když vykonávání programu je zastaveno.

[\GO3]

Group Output 3

Datový typ: signalgo

Proměnná signálu pro digitální skupinový procesní signál, který bude nastaven na nulu a dohlížen v datech restartu, když vykonávání programu je zastaveno.

[\GO4]

Group Output 4

Datový typ: signalgo

Proměnná signálu pro digitální skupinový procesní signál, který bude nastaven na nulu a dohlížen v datech restartu, když vykonávání programu je zastaveno.

Alespoň jeden z volitelných parametrů D01, GO1 ... GO4 musí být použit.

ShadowDO

Shadow Digital Output

Datový typ: signaldo

Proměnná signálu pro digitální signál, který musí zrcadlit, jestli je proces aktivní podél dráhy robotu nebo nikoliv.

Tento signál nebude nastaven na nulu procesem TriggStopProc na STOP nebo QSTOP, ale jeho hodnoty budou zrcadleny v restartdata.

Vykonávání programu

Nastavení a vykonání TriggStopProc

TriggStopProc musí být volán od obou:

- událostní rutiny START nebo v jednotkové části programu (nastavit PP na main, pozastavit interní proces pro TriggStopProc)
- událostní rutiny POWERON (napájení vypnout, pozastavit interní proces pro TriggStopProc)

Interní jméno procesu pro TriggStopProc je stejné jako jméno signálu v argumentu ShadowDO. Jestliže je vykonáván TriggStopProc se stejným jménem signálu v argumentu ShadowDO dvakrát ze stejné nebo jiné programové úlohy, potom pouze naposledy vykonávaný TriggStopProc bude aktivní.

Vykonávání TriggStopProc pouze spouští dohled I/O signálů na STOP a QSTOP.

Zastavení programu STOP

Proces TriggStopProc se skládá z následujících kroků:

1 Čekat, až robot zůstane stát v klidu na dráze.

2 Uložit aktuální hodnotu (předhodnota podle restartdata) všech použitých procesních signálů. Nula nastavuje všechny použité procesní signály kromě ShadowDO.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.302 TriggStopProc - Generovat restartovací data pro trigg signály při zastavení
RobotWare - OS
Pokračování

3 Proveďte následující během dalšího časového slotu, asi 500 ms, jestliže některé procesní signály změní svoji hodnotu během tohoto času:
   - Uložte znovu aktuální hodnotu (pohodnota podle `restatdata`)
   - Nastavte signál na nulu kromě `ShadowDO`
   - Spočítejte počet hodnotových přechodů (boků) signálu `ShadowDO`
4 Aktualizujte určenou perzistentní proměnnou s daty restartu.

Nouzové zastavení (`QSTOP`)

Proces `TriggStopProc` se skládá z následujících kroků:

1 Proveďte další krok tak rychle, jak je to možné.
2 Uložit aktuální hodnotu (předhodnota podle `restartdata`) všech použitých procesních signálů. Nastavit na nulu všechny použité procesní signály kromě `ShadowDO`.
3 Proveďte následující během dalšího časového slotu, asi 500 ms, jestliže některý procesní signál změní svoji hodnotu během tohoto času:
   - Uložte znovu jeho aktuální hodnotu (pohodnota podle `restatdata`
   - Nastavte tento signál na nulu kromě `ShadowDO`
   - Spočítejte počet hodnotových přechodů (boků) signálu `ShadowDO`
4 Aktualizujte určenou perzistentní proměnnou s daty restartu.

Kritická oblast pro restart procesu

Jak servo robotu, tak i externí vybavení se zpožďují. Všechny instrukce v rodině `Trigg` jsou navrženy tak, aby všechny signály byly nastaveny na vhodná místa na dráze robotu, nezávisle na různých zpožděních v externím vybavení, aby bylo možné získat tak dobré výsledky procesu, jak je to možné. Z toho důvodu mohou být nastavení I/O signálů zpožděna mezi 0-80 ms interně v systému, poté, kdy robot stojí v klidu po zastavení programu (`STOP`) nebo po registraci nouzového zastavení (`QSTOP`). Z důvodu této nevýhody pro funkčnost restartu jsou předhodnota, pohodnota a stínové boky uvedeny v datech restartu.

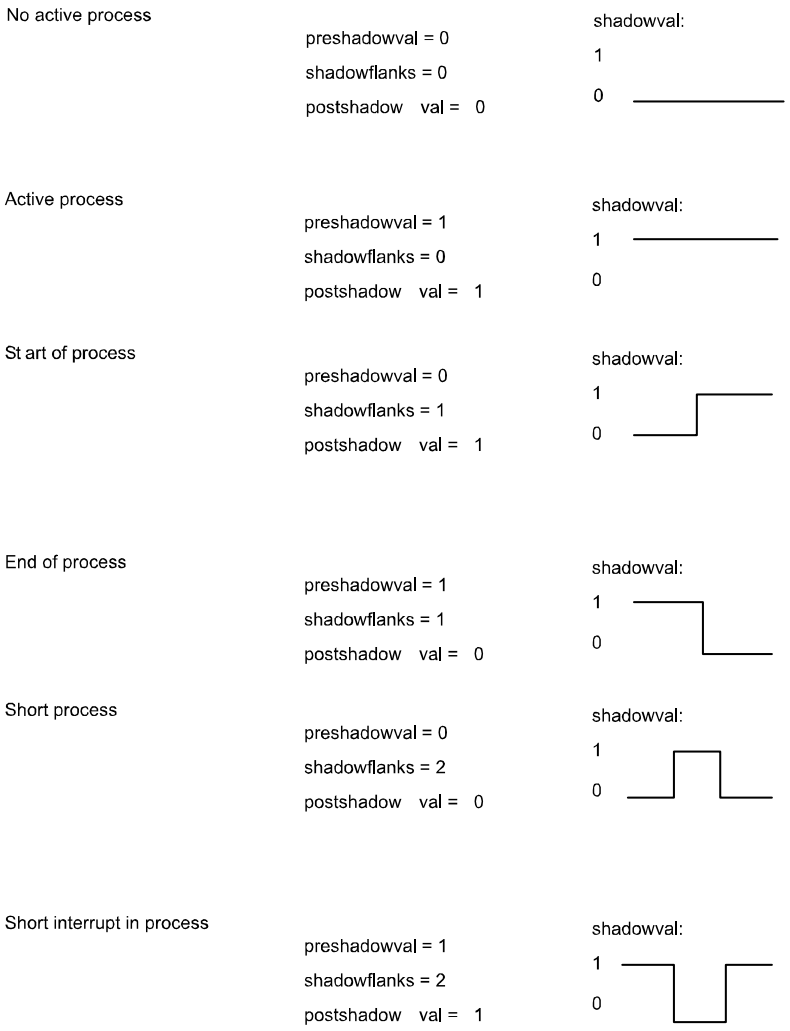
Jestliže tento kritický časový slot 0-80 ms koliduje s případy následujícího aplikačního procesu, potom je obtížné provést dobrý procesní restart:

- Na začátku aplikačního procesu
- Na konci aplikačního procesu
- Během krátkého aplikačního procesu
- Během krátkého přerušení v aplikačním procesu

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Obrázek dole ukazuje procesní fáze u STOP nebo QSTOP v kritickém časovém slotu 0-80 ms



xx0500002326

Provádění restartu

Restart procesních instrukcí (rutiny NOSTEPIN) podél dráhy robotu musí být proveden v událostní rutině RESTART.

Událostní rutina RESTART se může skládat z následujících kroků:

	Akce
1.	Po QSTOP je návrat na dráhu proveden při spuštění programu.
2.	Analyzovat data restartu z poslední STOP nebo QSTOP.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.302 TriggStopProc - Generovat restartovací data pro trigg signály při zastavení

RobotWare - OS

Pokračování

	Akce
3.	Určit strategii pro restart procesu z výsledku analýzy, jako je: • Proces je aktivní, proveďte restart procesu • Proces je neaktivní, neprovádějte restart procesu • Proveďte vhodné činnosti podle typu procesní aplikace: - Spuštění procesu - Konec procesu - Krátký proces - Krátké přerušení procesu
4.	Proveďte krok zpět na dráze.
5.	Pokračovat s výsledky programu při restartu pohybu.

Jestliže čekáme v jakékoliv událostní rutině `STOP` nebo `QSTOP`, až bude proces `TriggStopProc` připraven, např. s `WaitUntil (myproc.restartstop=TRUE), \MaxTime:=2;`, uživatel musí vždy resetovat příznak v událostní rutině `RESTART` např. s `myproc.restartstop:=FALSE`. Potom je restart připraven.

Řešení chyb

Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`. Při používání tohoto signálu je systémová proměnná `ERRNO` nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.

Jestliže neexistuje žádný kontakt s I/O jednotkou, systémová proměnná `ERRNO` je nastavena na `ERR_NORUNUNIT` a vykonávání pokračuje v chybovém handleru.

Omezení

Není podpora pro restart procesních instrukcí po výpadku napájení.

Syntaxe

```
TriggStopProc
  [ RestartRef ':=' ] < persistent (PERS) of restartdata>
  [ '\' DO1 ':=' < variable (VAR) of signaldo>
  [ '\' GO1 ':=' < variable (VAR) of signalgo> ]
  [ '\' GO2 ':=' < variable (VAR) of signalgo> ]
  [ '\' GO3 ':=' < variable (VAR) of signalgo> ]
  [ '\' GO4 ':=' < variable (VAR) of signalgo> ] ','
  [ ShadowDO ':=' ] < variable (VAR) of signaldo> ';'
```

Související informace

Pro informace o	Viz
Procesní instrukce	*TriggL - Lineární pohyby robotu s událostmi na str 839* *TriggC - Kruhový pohyb robotu s událostmi na str 796*
Data restartu	*restartdata - Data restartu pro signály trigg na str 1563*
Proveďte krok zpět na dráze	*StepBwdPath - Posunout zpět o jeden krok na dráze na str 720*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.302 TriggStopProc - Generovat restartovací data pro trigg signály při zastavení

RobotWare - OS

Pokračování

Pro informace o	Viz
Advanced RAPID	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.303 TryInt - Otestujte, jestli je datový objekt platným celým číslem.

Použití

`TryInt` se používá pro testování, jestli je daný datový objekt platným celým číslem.

Základní příklady

Následující příklady názorně ukazují instrukci `TryInt`:

Příklad 1

```
VAR num myint := 4;
...
TryInt myint;
```

Hodnota `myint` bude vyhodnocena a jelikož `4` je platné celé číslo, vykonávání programu pokračuje.

Příklad 2

```
VAR dnum mydnum := 20000000;
...
TryInt mydnum;
```

Hodnota `mydnum` bude vyhodnocena a jelikož `20000000` je platné celé číslo dnum, vykonávání programu pokračuje.

Příklad 3

```
VAR num myint := 5.2;
...
TryInt myint;
...
ERROR
  IF ERRNO = ERR_INT_NOTVAL THEN
    myint := Round(myint);
    RETRY;
  ENDIF
```

Hodnota `myint` bude vyhodnocena a jelikož `5.2` není platné celé číslo, bude pozvednuta chyba. V chybovém handleru bude `myint` zaokrouhleno na 5 a instrukce `TryInt` je vykonána ještě jednou.

Argumenty

```
TryInt DataObj | DataObj2
```

`DataObj`

Data Object

Datový typ: `num`

Datový objekt k otestování, jestli je platným celým číslem.

`DataObj2`

Data Object 2

Datový typ: `dnum`

Datový objekt k otestování, jestli je platným celým číslem.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Daný datový objekt je testován:

- Jestliže je platným celým číslem, vykonávání pokračuje s další instrukcí.
- Jestliže není platným celým číslem, vykonávání pokračuje v chybovém handleru v aktuální proceduře.

Řešení chyb

Jestliže `DataObj` obsahuje desetinnou hodnotu, potom bude proměnná `ERRNO` nastavena na `ERR_INT_NOTVAL`.

Jestliže hodnota `DataObj` je větší nebo menší než rozsah hodnoty celého čísla datového typu `num`, potom bude proměnná `ERRNO` nastavena na `ERR_INT_MAXVAL`.

Jestliže hodnota `DataObj2` je větší nebo menší než rozsah hodnoty celého čísla datového typu `dnum`, potom bude proměnná `ERRNO` nastavena na `ERR_INT_MAXVAL`.

Tyto chyby mohou být zpracovány v chybovém handleru.

Všimněte si, že hodnota `3.0` je vyhodnocena jako celé číslo, jelikož `.0` může být ignorována.

Syntaxe

```
TryInt
  [ DataObj ‘:=’ ] < expression (IN) of num>
  | [ DataObj2 ‘:=’ ] < expression (IN) of dnum>’ ;’
```

Související informace

Pro informace o	Viz
Datový typ num	*num - Numerické hodnoty na str 1538*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.304 TRYNEXT - Přeskočí instrukci, která způsobila chybu

Použití

Instrukce The TRYNEXT se používá pro obnovení vykonávání po chybě a začíná instrukcí následující po instrukci, která způsobila chybu.

Základní příklady

Následující příklad názorně ukazuje instrukci `TryNext`:

Příklad 1

```
reg2 := reg3/reg4;
...
ERROR
  IF ERRNO = ERR_DIVZERO THEN
    reg2:=0;
    TRYNEXT;
  ENDIF
```

Byl proveden pokus o rozdělení `reg3` a `reg4`. Jestliže `reg4` je rovný `0` (dělení nulou), potom je proveden skok k chybovém handleru, kde je `reg2` přidělen k `0`. Instrukce `TRYNEXT` se potom použije pro pokračování s další instrukcí.

Vykonávání programu

Vykonávání programu pokračuje s instrukcí následující po instrukci, která způsobila chybu.

Omezení

Instrukce může existovat pouze v chybovém handleru rutiny.

Syntaxe

```
TRYNEXT';'
```

Související informace

Pro informace o	Viz
Obslužné programy pro řešení chyb	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.305 TuneReset - Resetování ladění serva

Použití

`TuneReset` se používá pro reset dynamického chování všech os robotu a externích mechanických jednotek na jejich normální hodnoty.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `TuneReset`:

Příklad 1

```
TuneReset;
```

Resetování ladicích hodnot pro všechny osy na 100 %.

Vykonávání programu

Ladicí hodnoty pro všechny osy byly resetovány na 100 %.

Výchozí ladicí hodnoty serva pro všechny osy jsou automaticky nastaveny vykonáním instrukce `TuneReset`

- při **Restartu**.
- když je nový program načten.
- při spuštění provedení programu od začátku.

Syntaxe

```
TuneReset ';'
```

Související informace

Pro informace o	Viz
Ladění serv	*TuneServo - Ladění serv na str 886*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.306 TuneServo - Ladění serv

Použití

`TuneServo` se používá pro ladění dynamického chování samostatných os na robotu.

U většiny aplikací není nezbytné používat `TuneServo`, ale u některých aplikací je třeba `TuneServo`, abychom získali požadovanou přesnost. Použití `TuneServo` může být v mnoha případech nahrazeno výběrem předdefinovaného *Motion Process Mode*, viz *Technická referenční příručka - Systémové parametry* nebo úpravou předdefinovaného *Motion Process Mode*.

U externích os může být `TuneServo` použito pro adaptaci zátěže.

Vyhněte se provádění příkazů `TuneServo` ve stejném čase, kdy se robot pohybuje. Výsledkem může být krátkodobý vysoký točivý moment, který způsobí indikaci chyby a zastavení.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

POZNÁMKA

Abychom docílili optimálního ladění, je zásadní používat správná zátěžová data. Zkontrolujte to před použitím `TuneServo`.

VAROVÁNÍ

Nesprávné použití `TuneServo` může způsobit oscilační pohyby nebo točivé momenty, které mohou poškodit robot. Musíte to mít na paměti a být opatrní při používání `TuneServo`.

Popis

Snížit překmity a vibrace - `TUNE_DF`

`TUNE_DF` se může používat pro úpravu předvídaného kmitočtu mechanické rezonance konkrétní osy. Ladicí hodnota 95 % snižuje rezonanční kmitočet o 5 %. Nejobvyklejším použitím `TUNE_DF` je kompenzace vzniku neadekvátní tuhosti, tj. flexibilního vzniku. V tomto případě se ladicí hodnota pro osu 1 a 2 snižuje typicky na hodnotu mezi 80 % a 99 %.

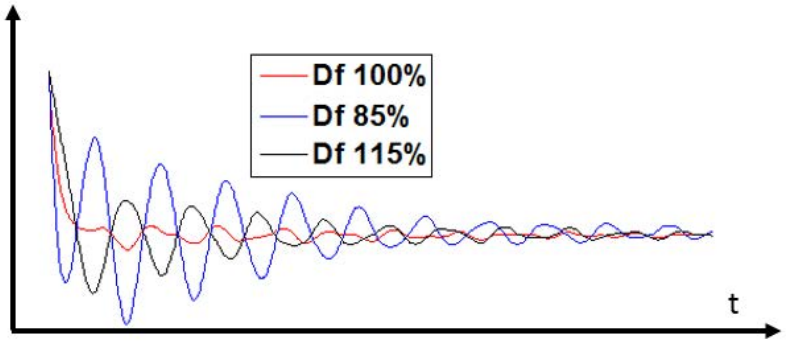
Použití `TUNE_DF` pro osy 3 - 6 je vzácné a normálně se nedoporučuje. Výjimkou je ladění os 4 - 6 kvůli kompenzaci rezonančního kmitočtu rozšířené flexibilní užitečné zátěže.

Správně upravené, nepříliš vysoké a nepříliš nízké `TUNE_DF` snižuje překmity a vibrace. Buďte opatrní při úpravě `TUNE_DF`, jelikož příliš vysoká nebo příliš nízká

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

ladicí hodnota může výrazně poškodit pohyb. Jeden příklad je uveden na obrázku dole. V tomto případě dává nejlepší výsledek ladicí hodnota 100 %.



xx1400001280

Ladicí hodnota může být automaticky optimalizována použitím TuneMaster, což se doporučuje.

U ručního ladění je příkladem část kódu RAPID pro ladění osy 1:

```
MoveAbsJ [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
     v200, fine, myTool;
FOR DF FROM 80 TO 100 STEP 5 DO
  TuneServo ROB_1,1,DF\Type:=TUNE_DF;
  MoveAbsJ [[2,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
       vmax, fine, myTool;
  WaitTime 1;
  MoveAbsJ [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
       vmax, fine, myTool;
  WaitTime 1;
ENDFOR
TuneReset;
```

Zde je ladicí hodnota měněna v krocích 5 %, může být použito také 2 %. Všimněte si, že pohyb by měl být krátký, typickou hodnotou jsou 2 stupně. Robot by měl být polohován do typické pozice pracovní oblasti. Vizuální prohlídkou by měla být zvolena ladicí hodnota, která minimalizuje překmity a vibrace.

Překmity a vibrace je také možné omezit snížením ladicí hodnoty pro `TUNE_DH` nebo snížením zrychlení pomocí `AccSet`. V mnoha případech je to nejlepší řešení. Nicméně, jestliže problém může být vyřešen pomocí `TUNE_DF`, doba cyklu není ovlivněna a použití `TUNE_DF` je tedy nejlepším řešením.

U robotů, kde je dostupný *Mounting Stiffness Factor*, viz *Motion Process Mode* v*Technická referenční příručka - Systémové parametry*, použití *Mounting Stiffness Factor* pro kompenzaci flexibilního vzniku nahrazuje použití `TUNE_DF`.

Snížit překmity a vibrace - `TUNE_DH`

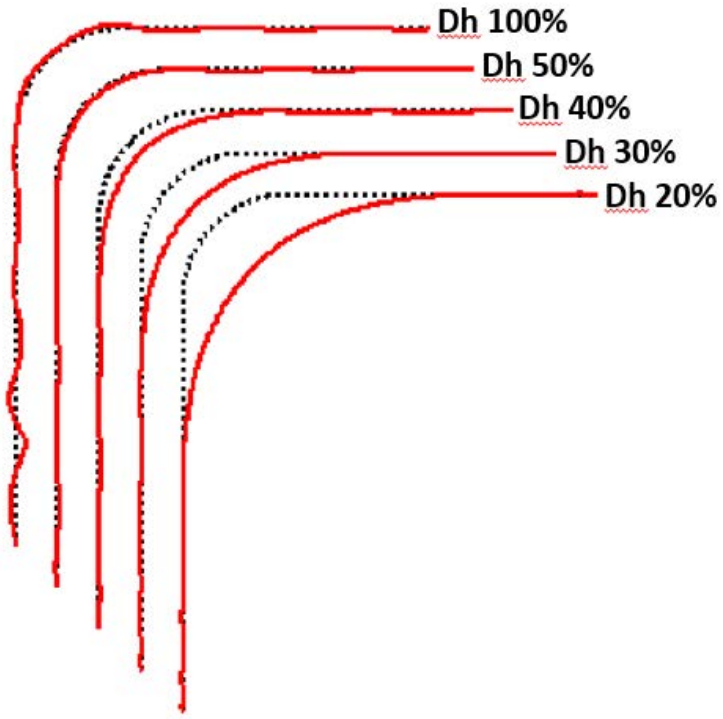
`TUNE_DH` může být použito pro zvýšení hladkosti dráhy robotu úpravou efektivní šířky pásma systému. Ladicí hodnota může být pouze snížena a hodnoty nad 100 % neovlivní pohyby. Ladicí hodnota menší než 100 % zmenšuje šířku pásma a zvyšuje hladkost, tedy snižuje překmity a vibrace.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

TUNE_DH pouze prodlužuje dobu cyklu v jemných bodech, jelikož snižování zrychlení prodlužuje dobu cyklu podél celé dráhy robotu. Proto použití TUNE_DH může být velmi účinnou cestou u doby cyklu jak snížit vibrace a překmity ve srovnání se snižováním zrychlení pomocí instrukce AccSet. Při vysoké rychlosti budou znatelné větší rohové zóny než naprogramované při použití TUNE_DH. Tedy, použití TUNE_DH snižuje chyby dráhy způsobené vibracemi, ale zvyšuje chyby dráhy při vysoké rychlosti převzetím zkratek do rohových zón. Zkratky se budou zvyšovat se snižující se ladicí hodnotou a zvyšující se rychlostí. Jestliže tyto zkratky nejsou přijatelné, doporučuje se AccSet namísto TUNE_DH.

Obrázek dole ukazuje účinek snížené ladicí hodnoty, a že nežádoucí vibrace mohou být odstraněny řádnou ladicí hodnotou. U menších ladicích hodnot se zkratka v rohové zóně stává znatelnou.



xx1400001281

Je dostatečné vykonávat instrukci TuneServo s argumentem \Type:=TUNE_DH pro jednu osu. Všechny osy ve stejné mechanické jednotce automaticky získají stejnou ladicí hodnotu.

Příklady:

- Řezání s TCP rychlostmi až 300 mm/s. Ladicí hodnota 50 % snižuje nežádoucí vibrace.

  Toto je někdy kombinováno s AccSet, např. AccSet 50,100;.
- Manipulace s materiálem při vysoké rychlosti, Ladicí hodnota 15 % snižuje nežádoucí vibrace.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

UPOZORNĚNÍ

Nikdy neměňte ladicí hodnotu, když robot je v pohybu, a buďte opatrní při používání malých ladicích hodnot (méně než 30 %), jelikož robot bude brát zkratky v rohových zónách.

Pouze pro vnitřní potřebu ABB - `TUNE_DK, TUNE_DL, TUNE_DG, TUNE_DI`

VAROVÁNÍ

Pouze pro vnitřní potřebu ABB. Nepoužívejte tyto typy ladění. Nesprávné použití může způsobit oscilační pohyby nebo krouticí momenty, které mohou poškodit robot.

Ladění externích os - `TUNE_KP`, `TUNE_KV`, `TUNE_TI`

Tyto typy ladění ovlivňují zisk kontroly pozice (kp), zisk kontroly rychlosti (kv) a dobu integrace kontroly rychlosti (ti) u externích os. Tyto jsou použity pro adaptaci externích os k různým setrvačnostem zátěže. Základní ladění externích os může být také zjednodušeno použitím těchto typů ladění.

Ladění os robotu - `TUNE_KP, TUNE_KV, TUNE_TI`

Tyto parametry se mohou používat pro změnu chování servo řadiče. `TUNE_KP` ovlivňuje ekvivalentní zisk pozičního řadiče, `TUNE_KV` ovlivňuje ekvivalentní zisk rychlostního řadiče, a `TUNE_TI` ovlivňuje integrální činnost řadiče.

Zvyšování ladicí hodnoty pro `TUNE_KV` zvyšuje tuhost serva robotu a může být užitečné v kontaktních aplikacích, jelikož celková tuhost systému robotu závisí na tuhosti serva a mechanické tuhosti. Zvýšená ladicí hodnota pro `TUNE_KV` také snižuje chyby dráhy při nízké rychlosti a může být užitečná v aplikacích řezání a svařování, kde je rychlost pod 100 mm/s. Typické ladicí hodnoty jsou 150% - 200%. Ladicí hodnota, která je příliš vysoká, způsobuje vibrace motoru a musí být vyloučena. Buďte vždy opatrní a pozorujte zvýšený hluk motoru při upravování `TUNE_KV` a nepoužívejte vyšší ladicí hodnoty, než je třeba ke splnění požadavků aplikace. Příliš vysoká ladicí hodnota může také zvyšovat vibrace vzhledem k mechanickým rezonancím.

Zvýšená ladicí hodnota pro `TUNE_KP` a snížená ladicí hodnota pro `TUNE_TI` zvyšuje tuhost serva a snižuje chyby dráhy při nízkých rychlostech v oblasti nízkých kmitočtů. Typické ladicí hodnoty pro `TUNE_KP` jsou 150% - 300%, a pro `TUNE_TI` je to 20% - 50%. Ve většině případů je `TUNE_KV` nejdůležitějším parametrem a `TUNE_KP` a `TUNE_TI` nepotřebují seřizování. Příliš vysoká ladicí hodnota pro `TUNE_KP` nebo příliš nízká ladicí hodnota pro `TUNE_TI` mohou také zvýšit vibrace vzhledem k mechanickým rezonancím.

Příklad:

- Robot v aplikaci začišťování potřebuje vyšší tuhost serva ke snížení chyb dráhy. `TUNE_KV` 175%, `TUNE_KP` 250%, a `TUNE_TI` 30%.

  Toto je často kombinováno s `AccSet`, např. `AccSet 30,100;`.

Pokračování na další straně

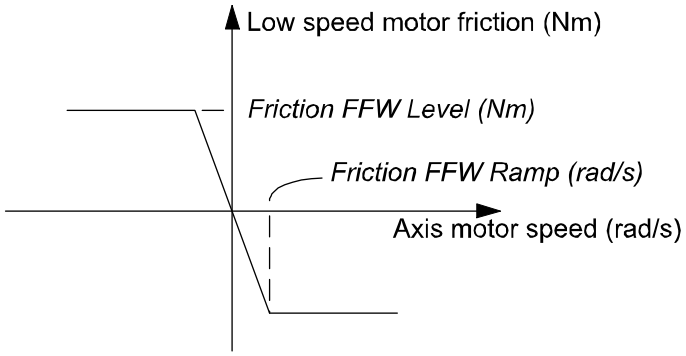
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.306 TuneServo - Ladění serv
RobotWare - OS
Pokračování

Kompenzace tření - TUNE_FRIC_LEV, TUNE_FRIC_RAMP

Tyto typy ladění se mohou používat k omezení chyb dráhy robotu způsobených třením a nadměrnou vůlí při nízkých rychlostech (10 - 200 mm/s). Tyto chyby dráhy se objevují, když osa robotu mění směr pohybu. Aktivujte kompenzaci tření pro osu nastavením systémového parametru Motion/Control Parameters/Friction FFW On na Yes.

Model tření je konstantní úroveň s opačným znaménkem směru rychlosti osy. *Friction FFW Level (Nm)* je absolutní úroveň tření při (nízkých) rychlostech a je větší než *Friction FFW Ramp (rad/s)*. Viz obrázek dole, kde je vidět model tření.



xx0500002188

TUNE_FRIC_LEV potlačuje hodnotu systémového parametru *Friction FFW Level*.

Ladění *Friction FFW Level* (pomocí TUNE_FRIC_LEV) pro každou osu robotu může významně zlepšit přesnost dráhy robotu v rychlostním rozsahu 20 - 100 mm/s. U větších robotů (zvláště v rodině IRB6400) bude účinek minimální, jelikož těmto robotů dominují jiné zdroje traťových chyb.

TUNE_FRIC_RAMP potlačuje hodnotu systémového parametru *Friction FFW Ramp*. Ve většině případů není potřeba ladit *Friction FFW Ramp*. Výchozí nastavení bude dostatečné.

Laďte vždy jen jednu osu. Ladicí hodnotu měňte v malých krocích a najděte úroveň, která minimalizuje chybu dráhy robotu v pozicích na dráze, kde tato konkrétní osa mění směr pohybu. Opakujte stejnou proceduru pro další osu a tak dále.

Konečné ladicí hodnoty mohou být přeneseny do systémových parametrů. Příklad:

Friction FFW Level = 1. Konečná ladicí hodnota (TUNE_FRIC_LEV) = 150%.

Nastavte Friction FFW Level = 1,5 a ladicí hodnota = 100% (výchozí hodnota), což je ekvivalent.

Argumenty

```
TuneServo MecUnit Axis TuneValue [\Type]
```

MecUnit

Mechanical Unit

Datový typ: mecunit

Jméno mechanické jednotky.

Axis

Datový typ: num

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Číslo aktuální osy pro mechanickou jednotku (1 - 6).

`TuneValue`

Datový typ: `num`

Ladicí hodnota v procentech (1 - 500). 100% je normální hodnota.

`[ \Type ]`

Datový typ: `tunetype`

Typ ladění serva. Dostupné typy jsou `TUNE_DF`, `TUNE_KP`, `TUNE_KV`, `TUNE_TI`, `TUNE_FRIC_LEV`, `TUNE_FRIC_RAMP`, `TUNE_DG`, `TUNE_DH`, `TUNE_DI`. Typ `TUNE_DK` a `TUNE_DL` pouze pro vnitřní potřebu ABB.

Tento argument je možné vypustit při použití ladicího typu `TUNE_DF`.

Základní příklady

Následující příklad názorně ukazuje instrukci `TuneServo`:

Příklad 1

```
TuneServo MHA160R1, 1, 110 \Type:= TUNE_KP;
```

Aktivace ladicího typu `TUNE_KP` s ladicí hodnotou `110`% na ose `1` v mechanické jednotce `MHA160R1`.

Vykonávání programu

Určený ladicí typ a ladicí hodnota jsou aktivovány pro určenou osu. Tato hodnota je použitelná pro všechny pohyby až do naprogramování nové hodnoty pro aktuální osu nebo až do resetování ladicích typů a hodnot pro všechny osy pomocí instrukce `TuneReset`.

Výchozí ladicí hodnoty serva pro všechny osy jsou automaticky nastaveny vykonáním instrukce `TuneReset`

- při **Restartu**.
- když je nový program načten.
- při spuštění provedení programu od začátku.

Omezení

Jakékoliv ladění aktivního serva je při výpadku napájení vždy nastaveno při restartu.

Toto omezení je možné ošetřit v uživatelském programu při restartu po výpadku napájení.

Syntaxe

```
TuneServo
  [MecUnit ':=' ] < variable (VAR) of mecunit> ','
  [Axis ':=' ] < expression (IN) of num> ','
  [TuneValue ':=' ] < expression (IN) of num>
  ['\' Type ':=' <expression (IN) of tunetype>] ';'
```

Související informace

Pro informace o	Viz
Ostatní nastavení pohybu	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.306 TuneServo - Ladění serv

RobotWare - OS

Pokračování

Pro informace o	Viz
Typy ladění serva	*tunetype - Typ servo ladění na str 1630*
Resetovat ladění všech serv	*TuneReset - Resetování ladění serva na str 885*
MotionProcessModeSet - Nastavit režim pohybového procesu.	*MotionProcessModeSet - Nastavit režim pohybového procesu na str 347*
Ladění externích os	*Application manual - Additional axes and stand alone controller*
Kompenzace tření	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.307 UIMsgBox - Základní typ dialogového boxu uživatelských zpráv

Použití

`UIMsgBox` (*User Interaction Message Box*) se používá ke komunikaci s uživatelem systému robotu na dostupném uživatelském zařízení, jako je FlexPendant. Zpráva je napsána k operátorovi, který odpoví volbou tlačítka. Uživatelská volba je potom přenesena zpět do programu.

Základní příklady

Následující příklady názorně ukazují instrukci `UIMsgBox`:

Viz také *Další příklady na str 898*.

Příklad 1

```
UIMsgBox "Continue the program ?";
```

Je zobrazena zpráva `"Continue the program ?"`. Program pokračuje, když uživatel stiskne výchozí tlačítko OK.

Příklad 2

```
VAR btnres answer;
...
UIMsgBox
  \Header:="UIMsgBox Header",
  "Message Line 1"
  \MsgLine2:="Message Line 2"
  \MsgLine3:="Message Line 3"
  \MsgLine4:="Message Line 4"
  \MsgLine5:="Message Line 5"
  \Buttons:=btnOKCancel
  \Icon:=iconInfo
  \Result:=answer;
IF answer = resOK my_proc;
```

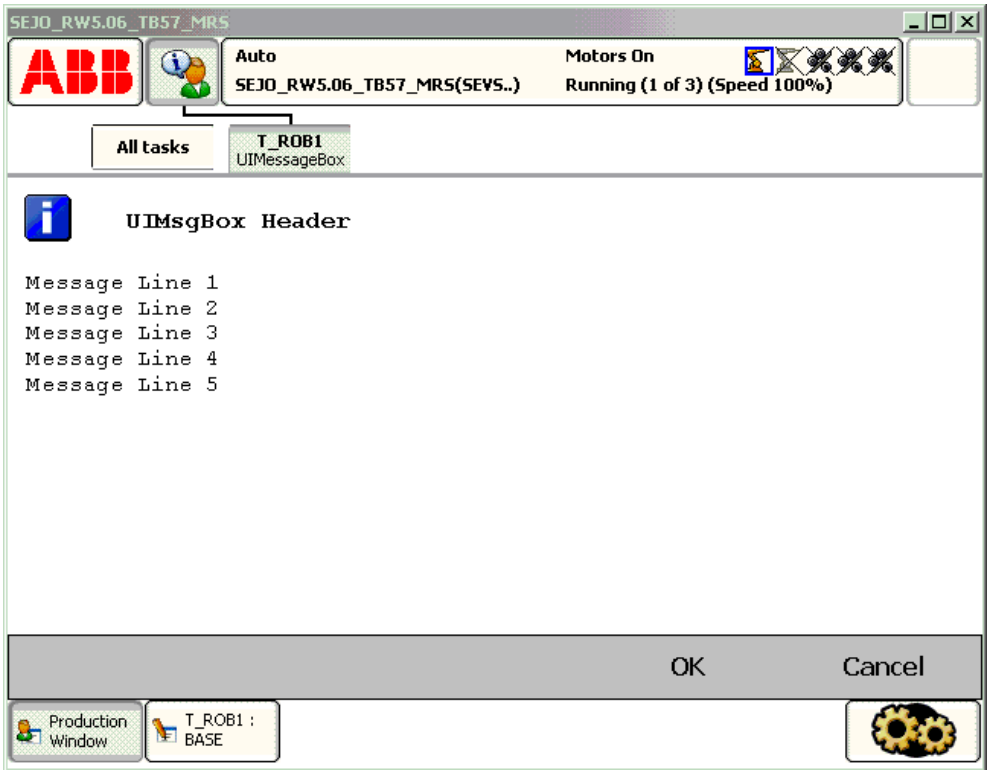
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.307 UIMsgBox - Základní typ dialogového boxu uživatelských zpráv

RobotWare - OS

Pokračování



xx0500002432

Shora uvedené okénko pro zprávu na displeji FlexPendantu obsahuje ikonu, hlavičku, řádku zprávy 1 až 5 a tlačítka. Vykonávání programu čeká, až je stisknuto OK nebo Cancel (Zrušit). Jinými slovy, pro `answer` bude přiděleno 1 (OK) nebo 5 (Cancel) podle toho, které tlačítko je stisknuto. Jestliže odpověď je OK, bude zavolán `my_proc`.

Všimněte si, že řádka zprávy 1 ... řádka zprávy 5 jsou zobrazeny na samostatných řádkách 1 až 5 (přepínač `\Wrap` není použit).

Argumenty

UIMsgBox [\Header] MsgLine1 [\MsgLine2] [\MsgLine3] [\MsgLine4] [\MsgLine5] [\Wrap] [\Buttons] [\Icon] [\Image] [\Result] [\MaxTime] [\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive] [\BreakFlag]

`[\Header]`

Datový typ: `string`

Text hlavičky, který bude zapsán v horní části okénka zprávy. Max. 40 znaků.

`MsgLine1`

Message Line 1

Datový typ: `string`

Textová řádka 1, která bude zapsána na displej. Max. 55 znaků.

`[\MsgLine2]`

Message Line 2

Datový typ: `string`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Dodatečná textová řádka 2, která bude zapsána na displej. Max. 55 znaků.

`[\MsgLine3]`

Message Line 3

Datový typ: `string`

Dodatečná textová řádka 3, která bude zapsána na displej. Max. 55 znaků.

`[\MsgLine4]`

Message Line 4

Datový typ: `string`

Dodatečná textová řádka 4, která bude zapsána na displej. Max. 55 znaků.

`[\MsgLine5]`

Message Line 5

Datový typ: `string`

Dodatečná textová řádka 5, která bude zapsána na displej. Max. 55 znaků.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny řetězce `MsgLine1 ... MsgLine5` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec zprávy *MsgLine1 ... MsgLine5* na samostatné řádce na displeji.

`[\Buttons]`

Datový typ: `buttondata`

Definuje tlačítka, která budou zobrazena. Může se použít pouze jedna z předdefinovaných kombinací tlačítek typu `buttondata`. Viz *Předdefinovaná data na str 897*.

Systém zobrazuje standardně tlačítko OK. (`\Buttons:=btn OK`).

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 897*.

Standardně žádná ikona.

`[\Image]`

Datový typ: `string`

Jméno obrázku, který by měl být použit. Chcete-li poslat své vlastní obrázky, musí být umístěny do adresáře `HOME:` v aktivním systému nebo přímo do aktivního systému.

Doporučuje se umístit soubory do adresáře `HOME:`, aby byly uloženy při provádění zálohování nebo obnovy.

Vyžaduje se **Restart** a potom FlexPendant načte obrázky.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.307 UIMsgBox - Základní typ dialogového boxu uživatelských zpráv
RobotWare - OS
Pokračování

Požadavkem na systém je, aby byl použit doplněk RobotWare *FlexPendant Interface*.

Obrázek, který má být zobrazen, může mít šířku 185 pixelů a výšku 300 pixelů. Jestliže je obrázek větší, pouze 185 * 300 pixelů obrázku bude vidět od horní části obrázku.

Není možné určovat přesnou hodnotu velikosti obrázků nebo počet obrázků, které mohou být načteny do FlexPendantu. Závisí to na velikosti ostatních souborů načtených do FlexPendantu. Vykonávání programu bude pouze pokračovat, jestliže je použit obrázek, který nebyl načten do FlexPendantu.

`[\Result]`

Datový typ: `btnres`

Proměnná, pro kterou je, v závislosti na tom, které tlačítko bylo stisknuto, vrácena numerická hodnota 0..7. Pro test uživatelského výběru se může použít jen jedna z předdefinovaných konstant typu `btnres`. Viz *Předdefinovaná data na str 897*.

Jestliže kterýkoliv typ přerušení systému, jako je `\MaxTime`, `\DIBreak` nebo `\DOBreak`, nebo jestliže `\Buttons:=btnNone, resUnkwn` je rovný 0, je vrácen.

`[\MaxTime]`

Datový typ: `num`

Max množství času v sekundách, kdy vykonávání programu čeká. Jestliže během této doby není zvoleno žádné tlačítko, program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_MAXTIME` může být použita pro testování, jestli uplynula max doba nebo nikoliv.

`[\DIBreak]`

Digital Input Break

Datový typ: `signaldi`

Digitální signál, který může přerušit dialog operátora. Jestliže není zvoleno žádné tlačítko, když signál je nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DIBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DIPassive]`

Digital Input Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DIBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DIBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DIBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\DOBreak]`

Digital Output Break

Datový typ: `signaldo`

Digitální výstupní signál, který může přerušit dialog operátora. Jestliže není zvoleno žádné tlačítko, když signál je nastaven na 1 (nebo již je 1), program pokračuje

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DOBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DOPassive]`

Digital Output Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DOBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DOBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DOBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\BreakFlag]`

Datový typ: `errnum`

Proměnná (před použitím je nastavena systémem na 0), která bude držte chybový kód, jestliže je použit `\MaxTime`, `\DIBreak`, or `\DOBreak`. Konstanty `ERR_TP_MAXTIME`, `ERR_TP_DIBREAK`, a `ERR_TP_DOBREAK` mohou být použity pro volbu příčiny. Jestliže tato volitelná proměnná není použita, potom bude vykonán chybový handler.

Vykonávání programu

Okénko pro zprávu s ikonou, hlavičkou, řádkami pro zprávu, obrázkem a tlačítky se zobrazí podle naprogramovaných argumentů. Vykonávání programu čeká, až uživatel zvolí jedno tlačítko nebo až je okénko zpráv přerušeno vypršením času nebo činností signálu. Uživatelská volba a důvod přerušení jsou převedeny zpět do programu.

Nové okénko pro zprávu na úrovni TRAP přebírá prioritu od okénka pro zprávu na základní úrovni.

Předdefinovaná data

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;

!Buttons:
  CONST buttondata btnNone := -1;
  CONST buttondata btnOK := 0;
  CONST buttondata btnAbrtRtryIgn := 1;
  CONST buttondata btnOKCancel := 2;
  CONST buttondata btnRetryCancel := 3;
  CONST buttondata btnYesNo := 4;
  CONST buttondata btnYesNoCancel := 5;

!Results:
  CONST btnres resUnkwn := 0;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.307 UIMsgBox - Základní typ dialogového boxu uživatelských zpráv
RobotWare - OS
Pokračování

```
CONST btnres resOK := 1;
CONST btnres resAbort := 2;
CONST btnres resRetry := 3;
CONST btnres resIgnore := 4;
CONST btnres resCancel := 5;
CONST btnres resYes := 6;
CONST btnres resNo := 7;
```

Další příklady

Více příkladů jak používat instrukci `UIMsgBox` je názorně uvedeno dole.

Příklad 1

```
VAR errnum err_var;
...
UIMsgBox \Header:= "Example 1", "Waiting for a break condition..."
     \Buttons:=btnNone \Icon:=iconInfo \MaxTime:=60 \DIBreak:=di5
     \BreakFlag:=err_var;

TEST err_var
CASE ERR_TP_MAXTIME:
  ! Time out break, max time 60 seconds has elapsed
CASE ERR_TP_DIBREAK:
  ! Input signal break, signal di5 has been set to 1
DEFAULT:
  ! Not such case defined
ENDTEST
```

Okénko pro zprávu je zobrazeno do splnění podmínky přerušení. Operátor nemůže odpovědět nebo odstranit okénko, protože `btnNone` je nastaven pro argument `\Buttons`. Okénko pro zprávu je odstraněno, když `di5` je nastaven na 1 nebo po vypršení času (po `60` sekundách).

Příklad 2

```
VAR errnum err_var;
...
UIMsgBox \Header:= "Example 2", "Waiting for a break condition..."
     \Buttons:=btnNone \Icon:=iconInfo \MaxTime:=60 \DIBreak:=di5
     \DIPassive \BreakFlag:=err_var;

TEST err_var
CASE ERR_TP_MAXTIME:
  ! Time out break, max time 60 seconds has elapsed
CASE ERR_TP_DIBREAK:
  ! Input signal break, signal di5 has been set to 0
DEFAULT:
  ! Not such case defined
ENDTEST
```

Okénko pro zprávu je zobrazeno do splnění podmínky přerušení. Operátor nemůže odpovědět nebo odstranit okénko, protože `btnNone` je nastaven pro argument `\Buttons`. Okénko pro zprávu je odstraněno, když `di5` je nastaven na 0 nebo po vypršení času (po `60` sekundách).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Jestliže parametr \BreakFlag není použit, tyto situace mohou být potom ošetřeny chybovým handlerem:

- Jestliže se objevilo vypršení času (parametr \MaxTime) před vstupem od operátora, systémová proměnná ERRNO je nastavena na ERR_TP_MAXTIME a vykonávání pokračuje v chybovém handleru.
- Jestliže je nastaven digitální vstup (parametr \DIBreak) před vstupem operátora, systémová proměnná ERRNO je nastavena na ERR_TP_DIBREAK a vykonávání pokračuje v chybovém handleru.
- Jestliže digitální výstup nastaven (parametr \DOBreak) před vstupem od operátora, systémová proměnná ERRNO je nastavena na ERR_TP_DOBREAK a vykonávání pokračuje v chybovém handleru.

Tato situace může být potom ošetřena chybovým handlerem:

- Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO, ERRNO je nastavena na ERR_NO_ALIASIO_DEF a vykonávání pokračuje v chybovém handleru.
- Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná ERRNO je nastavena na ERR_TP_NO_CLIENT a vykonávání pokračuje v chybovém handleru.

Omezení

Vyhněte se používání příliš malých hodnot pro parametr vypršení času \MaxTime, když je často vykonáván UIMsgBox, např. ve smyčce. Výsledkem může být nepředvídané chování činnosti systému, jak je zpomalení odezvy FlexPendantu.

Syntaxe

```
UIMsgBox
  [´\´Header´:=´ <expression (IN) of string>´,´]
  [MsgLine1´:=´] <expression (IN) of string>
  [´\´MsgLine2´:=´<expression (IN) of string>]
  [´\´MsgLine3´:=´<expression (IN) of string>]
  [´\´MsgLine4´:=´<expression (IN) of string>]
  [´\´MsgLine5´:=´<expression (IN) of string>]
  [´\´Wrap]
  [´\´Buttons´:=´ <expression (IN) of buttondata>]
  [´\´Icon´:=´ <expression (IN) of icondata>]
  [‘\’Image´:=´<expression (IN) of string>]
  [´\´Result´:=´< var or pers (INOUT) of btnres>]
  [´\´MaxTime´:=´ <expression (IN) of num>]
  [´\´DIBreak´:=´ <variable (VAR) of signaldi>]
  [´\´DIPassive]
  [´\´DOBreak´:=´ <variable (VAR) of signaldo>]
  [´\´DOPassive]
  [´\´BreakFlag´:=´ <var or pers (INOUT) of errnum>]´;´
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.307 UIMsgBox - Základní typ dialogového boxu uživatelských zpráv

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Data ikony displeje	*icondata - Data ikony displeje na str 1512*
Data tlačítka	*buttondata - Data tlačítka na str 1444*
Výsledková data tlačítka	*btnres - Výsledková data tlačítka na str 1441*
Pokročilý typ boxu zpráv uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*
Numerický vstup uživatelské interakce	*UINumEntry - Uživatelský číselný vstup na str 1417*
Numerické ladění uživatelské interakce	*UINumTune - Ladění uživatelských čísel na str 1423*
Alfa vstup uživatelské interakce	*UIAlphaEntry - Uživatelský vstup na str 1382*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Systém připojen k FlexPendantu a tak dále.	*UIClientExist - Existence uživatelského klienta na str 1388*
Rozhraní FlexPendantu	*Specifikace produktu - Controller software IRC5*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.308 UIShow - Ukázka uživatelského rozhraní

Použití

UIShow (*User Interface Show*) se používá ke komunikaci s uživatelem systému robotu na dostupném uživatelském zařízení, jako je FlexPendant. S UIShow mohou být individuálně napsané aplikace a standardní aplikace spuštěny z programu RAPID.

Základní příklady

Následující příklady názorně ukazují instrukci UIShow:

Příklad 1 a příklad 2 pracují pouze když soubory TpsViewMyAppl.dll a TpsViewMyAppl.gtpu.dll jsou přítomny v adresáři HOME: a byl proveden **Restart**.

Příklad 1

```
CONST string Name:="TpsViewMyAppl.gtpu.dll";
CONST string Type:="ABB.Robotics.SDK.Views.TpsViewMyAppl";
CONST string Cmd1:="Init data string passed to the view";
CONST string Cmd2:="New init data string passed to the view";
PERS uishownum myinstance:=0;
VAR num mystatus:=0;
...
! Launch one view of my application MyAppl
UIShow Name, Type \InitCmd:=Cmd1 \InstanceID:=myinstance
     \Status:=mystatus;
! Update the view with new init command
UIShow Name, Type \InitCmd:=Cmd2 \InstanceID:=myinstance
     \Status:=mystatus;
```

Kód nahoře spustí náhled TpsViewMyAppl s init příkazem Cmd1, a potom aktualizuje náhled s Cmd2.

Příklad 2

```
CONST string Name:="TpsViewMyAppl.gtpu.dll";
CONST string Type:="ABB.Robotics.SDK.Views.TpsViewMyAppl";
CONST string Cmd1:="Init data string passed to the view";
CONST string Cmd2:="New init data string passed to the view";
PERS uishownum myinstance:=0;
VAR num mystatus:=0;
...
! Launch one view of my application MyAppl
UIShow Name, Type \InitCmd:=Cmd1 \Status:=mystatus;
! Launch another view of the application MyAppl
UIShow Name, Type \InitCmd:=Cmd2 \InstanceID:=myinstance
     \Status:=mystatus;
```

Kód nahoře spustí náhled TpsViewMyAppl s init příkazem Cmd1. Potom spustí jiný náhled s init příkazem Cmd2.

Příklad 3

```
CONST string Name:="tpsviewbackupandrestore.dll";
CONST string Type:="ABB.Robotics.Tps.Views.TpsViewBackupAndRestore";
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.308 UIShow - Ukázka uživatelského rozhraní
Pokračování

```
VAR num mystatus:=0;
...
UIShow Name, Type \Status:=mystatus;
```

Spustit standardní aplikaci Záloha a obnova.

Příklad 4

```
CONST string Name:="TpsViewPanel.gtpu.dll";
CONST string Type:="ABB.Robotics.SDK.Views.MainScreen";
PERS uishownum myinstance:=0;
VAR num mystatus:=0;
...
UIShow Name, Type \InstanceID:=myinstance \Status:=mystatus;
```

Spustit aplikaci vytvořenou s `ScreenMaker`.

Argumenty

```
UIShow AssemblyName TypeName [\InitCmd] [\InstanceId] [\Status]
     [\NoCloseBtn]
```

`AssemblyName`

Datový typ: `string`

Jméno soustavy, která obsahuje náhled.

`TypeName`

Datový typ: `string`

Toto je jméno náhledu (typ k vytvoření). Toto je plně kvalifikované jméno typu, tj. jeho namespace je zahrnuto.

`[\InitCmd]`

Init Command

Datový typ: `string`

Řetězec init dat poskytnutých k náhledu.

`[\InstanceId]`

Datový typ: `uishownum`

Parametr, který reprezentuje token použitý k identifikaci náhledu. Jestliže náhled je zobrazen po volání k `UIShow`, hodnota, která identifikuje pohled, je poskytnuta zpět. Tento token je potom možné použít v jiných voláních k `UIShow` kvůli aktivaci již běžícího náhledu. Jestliže hodnota identifikuje existující (běžící) náhled, potom bude náhled aktivován.Jestliže neexistuje, bude vytvořena nová instance. To znamená, že tento parametr se může použít k určení, jestli nová instance bude spuštěna nebo nikoliv. Jestliže jeho hodnota identifikuje již spuštěný náhled, tento náhled bude aktivován bez ohledu na hodnoty všech jiných parametrů. Doporučuje se používat jedinečnou proměnnou `InstanceId` pro každou novou aplikaci, která bude spuštěna s instrukcí `UIShow`.

Parametr musí být perzistentní proměnná a důvodem je, že tato proměnná by si měla podržet svoji hodnotu, i když je ukazatel programu posunut na main. Při vykonávání stejného `UIShow` jako dříve a používání stejné proměnné bude aktivován stejný náhled, jestliže je ještě otevřen. Jestliže náhled byl zavřen, bude spuštěn nový náhled.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

[\Status]

Datový typ: num

Status indikuje, jestli operace byla úspěšná nebo nikoliv. Všimněte si, že když je tento doplněk použit, bude vykonávání RAPID čekat na dokončení instrukce, tj. náhled je spuštěn.

Tento volitelný parametr se primárně používá pro odladňovací účely. (Viz *Ošetřování chyb*)

Status	Popis
0	OK
-1	Pro nový náhled nezbylo na FlexPendantu žádné místo. Na FlexPendantu může být otevřeno max. 6 náhledů ve stejnou dobu.
-2	Sestava nebyla nalezena, neexistuje
-3	Soubor byl nalezen, ale nemůže být načten
-4	Sestava existuje, ale nelze vytvořit žádnou novou instanci
-5	typename je neplatný pro tuto sestavu
-6	InstanceID neodpovídá sestavě k načtení

[\NoCloseBtn]

No Close Button

Datový typ: switch

NoCloseBtn deaktivuje tlačítko zavírání náhledu.

Vykonávání programu

Instrukce UIShow se používá pro spuštění individuálních aplikací na FlexPendantu. Aby bylo možné spustit individuální aplikace, sestavy musí být umístěny do adresáře HOME: v aktivním systému nebo přímo do aktivního systému nebo do přídavného doplňku. Doporučuje se umístit soubory do adresáře HOME:, aby byly uloženy, jestliže se provádí zálohování a obnova. Vyžaduje se **Restart** a potom FlexPendant načítá nové sestavy. Požadavkem na systém je používání doplňku RobotWare *FlexPendant Interface*.

Je také možné spustit standardní aplikace jako je Záloha a Obnova. Potom není žádný požadavek na doplněk RobotWare *FlexPendant Interface*.

Při používání parametru \Status bude vykonávání programu čekat na spuštění aplikace. Jestliže chyby v aplikaci nejsou ošetřeny, potom je dohlížen pouze výsledek spuštění. Bez parametru \Status je FlexPendant přikázán ke spuštění aplikace, ale není zde žádná kontrola, která by určila, jestli je spuštění možné nebo nikoliv.

Řešení chyb

Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná ERRNO je nastavena na ERR_TP_NO_CLIENT a vykonávání pokračuje v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je použit parametr `\Status`, tyto situace mohou být potom ošetřeny chybovým handlerem:

- Jestliže na FlexPendantu nezbylo žádné místo pro sestavu, systémová proměnná `ERRNO` je nastavena na `ERR_UISHOW_FULL` a vykonávání pokračuje v chybovém handleru. FlexPendant může mít otevřeno 6 náhledů ve stejnou dobu.
- Jestliže se něco jiného pokazilo při pokusu spustit náhled, systémová proměnná `ERRNO` je nastavena na `ERR_UISHOW_FATAL` a vykonávání pokračuje v chybovém handleru.

Omezení

Při používání instrukce `UIShow` pro spuštění individuálních aplikací je zde požadavek, aby systém byl vybaven doplňkem *FlexPendant Interface*.

Aplikace, které byly spuštěny s instrukcí `UIShow`, nepřežijí situace výpadku napájení. Událostní rutinu `POWER ON` je možné použít pro opakované nastavení aplikace.

Syntaxe

```
UIShow
  [AssemblyName ´:=´] < expression (IN) of string >’,’
  [TypeName ´:=´] < expression (IN) of string >’,’
  [’\’InitCmd’ :=’ < expression (IN) of string> ]
  [’\’InstanceId ’:=’ < persistent (PERS) of uishownum> ]
  [’\’Status ’:=’ < variable (VAR) of num> ]
  [’\’NoCloseBtn ]’;’
```

Související informace

Pro informace o	Viz
FlexPendant interface	*Specifikace produktu - Controller software IRC5*
Stavění individuálních aplikací pro FlexPendant	http://developercenter.robotstudio.com/
`uishownum`	*uishownum - Instance ID pro UIShow na str 1631*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.309 UnLoad - Stáhnout programový modul během provádění

Použití

`UnLoad` se používá pro stažení programového modulu z paměti programu během vykonávání.

Programový modul musel být předtím načten do paměti programu pomocí instrukcí `Load` nebo `StartLoad - WaitLoad`.

Základní příklady

Následující příklad názorně ukazuje instrukci `UnLoad`:

Viz také *Další příklady* dole.

Příklad 1

```
UnLoad diskhome \File:="PART_A.MOD";
```

`UnLoad` programový modul PART_A.MOD z paměti programu, kam byl předtím načten s `Load`. (Viz instrukce `Load`). `diskhome` je předdefinovaná řetězcová konstanta "HOME:".

Argumenty

```
UnLoad [\ErrIfChanged] | [\Save] FilePath [\File]
```

`[\ErrIfChanged]`

Datový typ: `switch`

Jestliže se použije tento argument a modul byl změněn po načtení do systému, potom instrukce vygeneruje kód obnovy po chybě `ERR_NOTSAVED`.

`[\Save]`

Datový typ: `switch`

Jestliže je použit tento argument, potom je programový modul uložen před zahájením stahování. Programový modul bude uložen na původní místo určené v instrukcích `Load` nebo `StartLoad`

`FilePath`

Datový typ: `string`

Cesta souboru a jméno souboru k souboru, který bude stažen z paměti programu. Cesta souboru a jméno souboru musí být stejné jako v předtím vykonané instrukci `Load` nebo `StartLoad`. Jméno souboru by mělo být vyřazeno, když je použit argument `\File` .

`[\File]`

Datový typ: `string`

Když je jméno souboru vyřazeno v argumentu `FilePath`, musí být definováno s tímto argumentem. Jméno souboru musí být stejné, jako v předtím vykonané instrukci `Load` nebo `StartLoad` .

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.309 UnLoad - Stáhnout programový modul během provádění

RobotWare - OS

Pokračování

Vykonávání programu

Aby bylo možné vykonat instrukci `UnLoad` v programu, instrukce `Load` nebo `StartLoad - WaitLoad` se stejnou cestou a jménem souboru musí být vykonána dříve v programu.

Vykonávání programu čeká na stažení programového modulu, potom vykonávání pokračuje další instrukcí.

Potom je programový modul stažen a zbytek programových modulů bude propojen.

Další informace najdete v instrukcích `Load` nebo `StartLoad-Waitload`.

Další příklady

Více příkladů jak používat instrukci `UnLoad` je názorně uvedeno dole.

Příklad 1

```
UnLoad "HOME:/DOORDIR/DOOR1.MOD";
```

`UnLoad` programový modul `DOOR1.MOD` z paměti programu, kam byl předtím načten.

Příklad 2

```
UnLoad "HOME:" \File:="DOORDIR/DOOR1.MOD";
```

Stejné jako v příkladu 1 nahoře, ale s jinou syntaxí.

Příklad 3

```
Unload \Save, "HOME:" \File:="DOORDIR/DOOR1.MOD";
```

Stejné, jako v příkladech 1 a 2 nahoře, ale ukládá programový modul před jeho stažením.

Omezení

Není dovoleno stahovat programový modul, který je ve fázi vykonávání (ukazatel programu v modulu).

TRAP rutiny, systémové I/O události a další programové úlohy se nemohou vykonávat během procesu stahování.

Vyloučit probíhající pohyby robotu během stahování.

Zastavení programu během vykonávání instrukce `UnLoad` má za výsledek zastavení ochrany s vypnutím motorů a chybovou zprávou "20025 Stop order timeout" na FlexPendantu.

Řešení chyb

Jestliže soubor v instrukci `UnLoad` nemůže být stažen kvůli probíhajícímu vykonávání v rámci modulu nebo špatné cestě (modul nebyl načten s `Load` nebo `StartLoad`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UNLOAD`.

Jestliže je použit argument `\ErrIfChanged` a modul byl změněn, vykonání této rutiny nastaví systémovou proměnnou `ERRNO` na `ERR_NOTSAVED`.

Tyto chyby mohou být potom zpracovány v chybovém handleru.

Syntaxe

```
UnLoad
  ['\'ErrIfChanged ','] | ['\'Save ',']
  [FilePath':=']<expression (IN) of string>
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
['\'File':=' <expression (IN) of string>]';'
```

Související informace

Pro informace o	Viz
Zkontrolovat reference programu	*CheckProgRef - Zkontrolovat reference programu na str 103*
Načíst programový modul	*Load - Načíst programový modul během provádění na str 328* *StartLoad - Načíst programový modul během vykonávání na str 703* *WaitLoad - Připojit načtený modul k úloze na str 947*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.310 UnpackRawBytes - Rozbalit data z rawbyte dat

Použití

`UnpackRawBytes` se používá k rozbalení obsahu kontejneru typu `rawbytes` do proměnných typu `byte`, `num`, `dnum` nebo `string`.

Základní příklady

Následující příklad názorně ukazuje instrukci `UnpackRawBytes`:

Příklad 1

```
VAR iodev io_device;
VAR rawbytes raw_data_out;
VAR rawbytes raw_data_in;
VAR num integer;
VAR dnum bigInt;
VAR num float;
VAR string string1;
VAR byte byte1;
VAR byte data1;

! Data packed in raw_data_out according to the protocol
...
Open "chan1:", io_device\Bin;
WriteRawBytes io_device, raw_data_out;
ReadRawBytes io_device, raw_data_in\Time := 1;
Close io_device;
```

Podle protokolu, který je znám programátorovi, je zpráva odeslána do zařízení `"chan1:"`. Potom je přečtena odpověď ze zařízení.

Odpověď obsahuje například následující:

počet bajtů:	obsah:
1-4	integer‘ 5’
5-8	float‘ 234.6’
9-25	řetězec "This is real fun!"
26	hex value‘ 4D’
27	ASCII kód 122, tj. ‘z’
28-36	integer‘ 4294967295’
37-40	integer‘ 4294967295’

```
UnpackRawBytes raw_data_in, 1, integer \IntX := DINT;
```

Obsah `integer` bude `5`.

```
UnpackRawBytes raw_data_in, 5, float \Float4;
```

Obsah `float` bude desetinné číslo `234.6`.

```
UnpackRawBytes raw_data_in, 9, string1 \ASCII:=17;
```

Obsah `string1` bude `"This is real fun!"`.

```
UnpackRawBytes raw_data_in, 26, byte1 \Hex1;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Obsah `byte1` **bude šestnáctkové** `'4D'`.

```
UnpackRawBytes raw_data_in, 27, data1 \ASCII:=1;
```

Obsah `data1` **bude** `122`, **ASCII kód pro "z".**

```
UnpackRawBytes raw_data_in, 28, bigInt \IntX := LINT;
```

Obsah `bigInt` **bude** `4294967295`.

```
UnpackRawBytes raw_data_in, 37, bigInt \IntX := UDINT;
```

Obsah `bigInt` **bude** `4294967295`.

Argumenty

```
UnpackRawBytes RawData [ \Network ] StartIndex Value [\Hex1 ]| [
     \IntX ] | [ \Float4 ] | [ \ASCII ]
```

`RawData`

Datový typ: `rawbytes`

Kontejner proměnné pro rozbalení dat.

`[ \Network ]`

Datový typ: `switch`

Indikuje, že integer a float bude rozbaleno z big endian (síťový příkaz) reprezentovaném v `RawData`**. ProfiBus a InterBus používají big endian.**

Bez tohoto přepínače bude integer a float rozbalen v little endian (nesíťový příkaz) reprezentaci od `RawData`**. DeviceNet používá little endian.**

Relevantní pouze společně s volitelným parametrem `\IntX - UINT`, `UDINT`, **ULINT**, **INT**, **DINT**, **LINT a** `\Float4`.

`StartIndex`

Datový typ: `num`

`StartIndex`, **mezi 1 a 1024, indikuje, kde začít rozbalovat data z** `RawData`.

`Value`

Datový typ: `anytype`

Proměnná obsahující data, která byla rozbalena z `RawData`.

Přípustné datové typy: `byte`, `num`, `dnum` **nebo** `string`. **Pole není možné používat.**

`[ \Hex1 ]`

Datový typ: `switch`

Data, která budou rozbalena a umístěna v `Value` **mají šestnáctkový formát v 1** `byte` **a budou převedena do desítkového formátu v proměnné** `byte`.

`[ \IntX ]`

Datový typ: `inttypes`

Data k rozbalení mají formát podle určené konstanty datového typu `inttypes`. **Data budou převedena do proměnné** `num` **nebo** `dnum` **obsahující celé číslo a uložena do** `Value`.

Viz část *Předdefinovaná data na str 910*.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.310 UnpackRawBytes - Rozbalit data z rawbyte dat

RobotWare - OS

Pokračování

`[ \Float4 ]`

Datový typ: `switch`

Data k rozbalení a umístění do Value mají float, 4 bajty, formát a budou převedena do proměnné `num` obsahující float.

`[ \ASCII ]`

Datový typ: `num`

Data k rozbalení a umístění do `Value` mají formát `byte` nebo `string`.

Jestliže `Value` je typu `byte`, potom budou data interpretována jako ASCII kód a převedena na formát `byte` (1 znak).

Jestliže `Value` je typu `string`, potom budou data uložena jako `string` (1...80 znaků). Řetězcová data nejsou zakončena NULOU v datech typu `rawbytes`.

Musí být naprogramován jeden z argumentů `\Hex1`, `\IntX`, `\Float4`, nebo `\ASCII`.

Jsou dovoleny následující kombinace:

Datový typ Value (Hodnota):	Dovolené volitelné parametry:
`num` *)	`\IntX`
`dnum` **)	`\IntX`
`num`	`\Float4`
`string`	`\ASCII:=n with n between 1 and 80`
`byte`	`\Hex1 \ASCII:=1`

*) Musí být celé číslo v rozsahu hodnoty zvolené symbolické konstanty `USINT`, `UINT`, `UDINT`, `SINT`, `INT` nebo `DINT`.

**) Musí být celé číslo v rozsahu hodnoty zvolené symbolické konstanty `USINT`, `UINT`, `UDINT`, `ULINT`, `SINT`, `INT`, `DINT` nebo `LINT`.

Vykonávání programu

Během vykonávání programu jsou data rozbalena z kontejneru typu `rawbytes` do proměnné typu `anytype`.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Předdefinovaná data

Následující symbolické konstanty datového typu `inttypes` jsou předdefinovány a mohou se používat k určení celého čísla v argumentu `\IntX`.

Symbolická konstanta	Konstantní hodnota	Formát celého čísla	Rozsah hodnoty celého čísla
`USINT`	1	Nepodepsané 1-bajtové celé číslo	0 ... 255
`UINT`	2	Nepodepsané 2-bajtové celé číslo	0 ... 65 535
`UDINT`	4	Nepodepsané 4-bajtové celé číslo	0 ... 8 388 608 *) 0 ... 4 294 967 295 ****)

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Symbolická konstanta	Konstantní hodnota	Formát celého čísla	Rozsah hodnoty celého čísla
ULINT	8	Nepodepsané 8-bajtové celé číslo	0 ... 4 503 599 627 370 496**)
SINT	- 1	Podepsané 1-bajtové celé číslo	- 128... 127
INT	- 2	Podepsané 2-bajtové celé číslo	- 32 768 ... 32 767
DINT	- 4	Podepsané 4-bajtové celé číslo	- 8 388 607 ... 8 388 608 *) -2 147 483 648 ... 2 147 483 647 ***)
LINT	- 8	Podepsané 8-bajtové celé číslo	- 4 503 599 627 370 496... 4 503 599 627 370 496 **)

*) RAPID omezení pro ukládání celého čísla v datovém typu num.

**) RAPID omezení pro ukládání celého čísla v datovém typu dnum.

***) Rozsah při používání proměnné dnum a inttype DINT.

****) Rozsah při používání proměnné dnum a inttype UDINT.

Syntaxe

```
UnpackRawBytes
  [RawData ´:=´ ] < variable (VAR) of rawbytes>
  [ ’\’ Network ] ´,´
  [StartIndex ´:=´ ] < expression (IN) of num> ´,´
  [Value ’:=’ ] < variable (VAR) of anytype>
  [ ’\’ Hex1 ]
  | [ ’\’ IntX’ :=’ < expression (IN) of inttypes>]
  | [’ \’ Float4 ]
  | [ ’\’ ASCII‘ :=’ < expression (IN) of num>] ’;’
```

Související informace

Pro informace o	Viz
Datarawbytes	*rawbytes - Data raw na str 1559*
Získat délku dat rawbytes	*RawBytesLen - Získat délku dat rawbytes na str 1278*
Vyčistit obsah dat rawbytes	*ClearRawBytes - Vyčistit obsahy rawbyte dat na str 118*
Kopírovat obsah dat rawbytes	*CopyRawBytes - Kopírovat obsah dat rawbytes na str 137*
Zabalit hlavičku DeviceNet do dat rawbytes	*PackDNHeader - Zabalit hlavičku DeviceNet do dat rawbytes na str 446*
Zabalit data do dat rawbytes	*PackRawBytes - Zabalit data do dat rawbytes na str 449*
Zapsat data rawbytes	*WriteRawBytes - Zapsat data rawbytes na str 994*
Načíst data rawbytes	*ReadRawBytes - Přečíst data rawbytes na str 530*
Rozbalit data z dat rawbytes	*UnpackRawBytes - Rozbalit data z rawbyte dat na str 908*
Funkce Bit/Bajt	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.310 UnpackRawBytes - Rozbalit data z rawbyte dat

RobotWare - OS

Pokračování

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.311 VelSet - Mění naprogramovanou rychlost

Použití

`VelSet` se používá ke zvýšení nebo snížení naprogramované rychlosti všech následných polohovacích instrukcí. Tato instrukce se také používá k maximalizaci rychlosti.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje instrukci `VelSet`:

Viz také *Další příklady na str 914*.

Příklad 1

```
VelSet 50, 800;
```

Všechny naprogramované rychlosti jsou sníženy na 50 % hodnoty v instrukci. Rychlost TCP nemá povolení překročit 800 mm/s.

Argumenty

```
VelSet Override Max
```

`Override`

Datový typ: `num`

Požadovaná rychlost jako procento naprogramované rychlosti. 100 % odpovídá naprogramované rychlosti.

`Max`

Datový typ: `num`

Maximální rychlost TCP v mm/s.

Vykonávání programu

Naprogramovaná rychlost všech následných polohovacích instrukcí je ovlivněna až do vykonání nové instrukce `VelSet`.

Argument `Override` ovlivňuje:

- Všechny rychlostní komponenty (TCP, orientace, otáčení a lineární externí osy) v `speeddata`.
- Potlačení naprogramované rychlosti v polohovací instrukci (argument `\V`).
- Načasované pohyby.

Argument `Override` neovlivňuje:

- Rychlost svařování v `welddata`.
- Rychlost prohřívání a plnění v `seamdata`.

Argument `Max` ovlivňuje pouze rychlost TCP.

Výchozí hodnoty pro `Override` a `Max` jsou 100% a `vmax.v_tcp` mm/s *). Tyto hodnoty se nastavují automaticky

- při používání restartovacího režimu **Reset RAPID**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.311 VelSet - Mění naprogramovanou rychlost

RobotWare - OS

Pokračování

- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

*) Max rychlost TCP pro použitý typ robotu a normální praktické hodnoty TCP. RAPID funkce `MaxRobSpeed` vrací stejnou hodnotu.

Další příklady

Více příkladů jak používat instrukci `VelSet` je názorně uvedeno dole.

Příklad 1

```
VelSet 50, 800;
MoveL p1, v1000, z10, tool1;
MoveL p2, v2000, z10, tool1;
MoveL p3, v1000\T:=5, z10, tool1;
```

Rychlost je 500 mm/s k bodu `p1` a `800` mm/s k `p2`. Trvání pohybu `10` sekund od `p2` k `p3`.

Omezení

Max rychlost není brána v úvahu, když je určen čas v polohovací instrukci.

Syntaxe

```
VelSet
  [ Override ´:=´ ] < expression (IN) of num > ´,´
  [ Max ´:=´ ] < expression (IN) of num > ´;´
```

Související informace

Pro informace o	Viz
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Max. rychlost TCP pro tento robot	*MaxRobSpeed - Max rychlost robotu na str 1222*
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.312 WaitAI - Čeká na nastavení hodnoty analogového vstupního signálu

Použití

`WaitAI` (*Wait Analog Input*) se používá k čekání na nastavení hodnoty analogového vstupního signálu.

Základní příklady

Následující příklady názorně ukazují instrukci `WaitAI`:

Příklad 1

```
WaitAI ai1, \GT, 5;
```

Vykonávání programu pokračuje, až když analogový vstup `ai1` má hodnotu větší než 5.

Příklad 2

```
WaitAI ai1, \LT, 5;
```

Vykonávání programu pokračuje, až když analogový vstup `ai1` má hodnotu menší než 5.

Argumenty

```
WaitAI Signal [\LT] | [\GT] Value [\MaxTime] [\ValueAtTimeout]
     [\Visualize] [\Header] [\Message] | [\MsgArray] [\Wrap]
     [\Icon] [\Image] [\VisualizeTime]
```

`Signal`

Datový typ: `signalai`

Jméno analogového výstupního signálu.

`[\LT]`

Less Than

Datový typ: `switch`

Při používání tohoto parametru instrukce `WaitAI` čeká, až hodnota analogového signálu bude menší než hodnota v `Value`.

`[\GT]`

Greater Than

Datový typ: `switch`

Při používání tohoto parametru instrukce `WaitAI` čeká, až hodnota analogového signálu bude větší než hodnota v `Value`.

`Value`

Datový typ: `num`

Požadovaná hodnota signálu.

`[\MaxTime]`

Maximum Time

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.312 WaitAI - Čeká na nastavení hodnoty analogového vstupního signálu
RobotWare - OS
Pokračování

Max povolená délka času čekání vyjádřená v sekundách. Jestliže tento čas vyprší před splněním podmínky, bude volán chybový handler (pokud existuje) s chybovým kódem ERR_WAIT_MAXTIME. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

[\ValueAtTimeout]

Datový typ: num

Jestliže vyprší čas instrukce, aktuální hodnota signálu bude uložena do této proměnné. Proměnná bude nastavena, pouze když systémová proměnná ERRNO je nastavena na ERR_WAIT_MAXTIME.

[\Visualize]

Datový typ: switch

Jestliže je zvoleno, aktivuje se vizualizace. Vizualizace se skládá z boxu zpráv s podmínkou, která nebyla splněna, ikony, hlavičky, řádek zprávy a obrázek je zobrazen podle naprogramovaných argumentů.

[\Header]

Datový typ: string

Text hlavičky je napsán v horní části boxu zpráv. Maximum je 40 znaků. Jestliže není použit žádný argument \Header, bude zobrazena výchozí zpráva.

[\Message]

Datový typ: string

Na displej bude napsána jedna textová řádka. Maximum je 50 znaků.

[\MsgArray]

(Message Array)

Datový typ: string

Několik textových řádek od pole, které budou napsány na displej. Může být použit pouze jeden z parametrů \Message nebo \MsgArray ve stejném čase.

Max prostor je 5 řádek s 50 znaky na každé z nich.

[\Wrap]

Datový typ: switch

Jestliže je tak zvoleno, všechny určené řetězce v argumentu \MsgArray budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu \MsgArray na samostatné řádce na displeji.

[\Icon]

Datový typ: icondata

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu icondata. Viz *Předdefinovaná data na str 1512*.

Výchozí nastavení, žádná ikona.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[\Image]`

Datový typ: `string`

Jméno obrázku, který by měl být použit. Chcete-li poslat své vlastní obrázky, musí být umístěny do adresáře *HOME:* v aktivním systému nebo přímo do aktivního systému.

Doporučuje se umístit soubory do adresáře *HOME:*, aby byly uloženy při provádění zálohování a obnovy.

Vyžaduje se Restart a potom FlexPendant načte obrázky.

Požadavkem na systém je, aby byl použit doplněk RobotWare *FlexPendant Interface*.

Obrázek, který má být zobrazen, může mít šířku 185 a výšku 300. Jestliže je obrázek větší, pouze 185 * 300 pixelů obrázku bude vidět od horní levé části obrázku.

Není možné určovat přesnou hodnotu velikosti obrázků nebo počet obrázků, které mohou být načteny do FlexPendantu. Závisí to na velikosti ostatních souborů načtených do FlexPendant. Vykonávání programu bude pouze pokračovat, jestliže je použit obrázek, který nebyl načten do FlexPendant.

`[\VisualizeTime]`

Datový typ: `num`

Čas čekání před tím, než by se měl objevit box zpráv na FlexPendant. Jestliže používáme argumenty `\VisualizeTime` a `\MaxTime`, čas použitý v argumentu `\MaxTime` musí být delší než čas použitý v argumentu `\VisualizeTime`.

Výchozí čas pro vizualizaci, jestliže nepoužíváme argument `\VisualizeTime` je 5 s. Minimální hodnota 1 s. Maximální hodnota nemá žádný limit. Rozlišení 0.001 s.

Vykonávání programu

Jestliže hodnota signálu je správná, když se vykonává instrukce, program jednoduše pokračuje s následující instrukcí.

Jestliže hodnota signálu je nesprávná, robot vstoupí do stavu čekání a program pokračuje, když se signál změní na správnou hodnotu. Změna je zjištěna přerušením, což dává rychlou odezvu (bez dotazování).

Když robot čeká, čas je dohlížen. Podle výchozího nastavení může robot čekat stále, ale max dobu čekání je možné určit volitelným argumentem `\MaxTime`. Jestliže tento max čas je překročen, je vyvolána chyba.

Jestliže vykonávání programu je zastaveno a později obnoveno, instrukce vyhodnotí aktuální hodnotu signálu. Jakákoliv změna během zastavení programu je odmítnuta.

V ručním režimu, po čekání delším než 3 s se objeví okénko s upozorněním a dotazem, jestli chcete simulovat instrukci. Jestliže nechcete, aby se okénko s upozorněním objevovalo, můžete nastavit systémový parametr *SimulateMenu* na NO, viz *Technická referenční příručka - Systémové parametry*, téma *Controller*, typ *General RAPID*.

Jestliže je použit přepínač `\Visualize`, box zpráv se zobrazí na FlexPendant podle naprogramovaných argumentů. Jestliže není použit žádný argument `\Header`,

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.312 WaitAI - Čeká na nastavení hodnoty analogového vstupního signálu
RobotWare - OS
Pokračování

zobrazí se výchozí text hlavičky. Když je vykonávání instrukce `WaitAI` připraveno, box zpráv bude odstraněn z FlexPendant.

Nový box zpráv na úrovni TRAP přebírá prioritu od boxu zpráv na základní úrovni.

Další příklady

Více příkladů instrukce `WaitAI` je názorně uvedeno dole.

Příklad 1

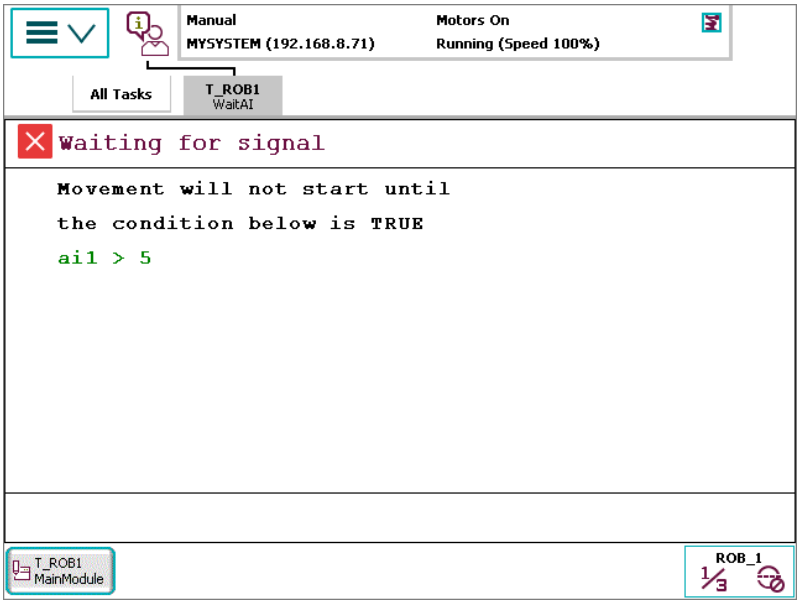
```
VAR num myvalattimeout:=0;
WaitAI ai1, \LT, 5 \MaxTime:=4 \ValueAtTimeout:=myvalattimeout;
ERROR
  IF ERRNO=ERR_WAIT_MAXTIME THEN
    TPWrite "Value of ai1 at timeout:" + ValToStr(myvalattimeout);
    TRYNEXT;
  ELSE
    ! No error recovery handling
  ENDIF
```

Vykonávání programu pokračuje pouze když `ai1` je méně než 5 nebo po vypršení času. Jestliže čas vypršel, hodnota signálu `ai1` při vypršení času může být zapsána bez dalšího čtení signálu.

Příklad 2

```
WaitAI ai1 \GT, 5 \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Jestliže podmínka není splněna, potom bude hlavička a zpráva určená ve volitelných argumentech `\Header` a `\MsgArray` zapsána na displej FlexPendantu společně s podmínkou, které není splněna.



xx1600000150

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_AO_LIM, jestliže naprogramovaný argument Value pro určený analogový vstupní signál Signal je mimo limity.

ERR_NO_ALIASIO_DEF jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.

ERR_NORUNUNIT jestliže není žádný kontakt s jednotkou I/O.

ERR_SIG_NOT_VALID jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

ERR_WAIT_MAXTIME, jestliže došlo k vypršení času (parametr \MaxTime) před změnou signálu na správnou hodnotu.

Syntaxe

```
WaitAI
  [ Signal ':=' ] < variable (VAR) of signalai> ´,´
  [ '\' LT] | [ '\' GT] ','
  [ Value ':=' ] < expression (IN) of num>
  ['\'MaxTime ´:='<expression (IN) of num>]
  [ '\'ValueAtTimeout' :=' < variable (VAR) of num >]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Čekání na splnění podmínky	*WaitUntil - Čeká na splnění podmínky na str 965*
Čekání na určený časový úsek	*WaitTime - Čeká danou dobu na str 963*
Čekání na nastavení nebo resetování analogového výstupu	*WaitAO - Čeká na nastavení hodnoty analogového výstupního signálu na str 920*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.313 WaitAO - Čeká na nastavení hodnoty analogového výstupního signálu

Použití

`WaitAO` (*Wait Analog Output*) se používá k čekání na nastavení hodnoty analogového výstupního signálu.

Základní příklady

Následující příklady názorně ukazují instrukci `WaitAO`:

Příklad 1

```
WaitAO ao1, \GT, 5;
```

Vykonávání programu pokračuje, až když analogový výstup `ao1` má hodnotu větší než 5.

Příklad 2

```
WaitAO ao1, \LT, 5;
```

Vykonávání programu pokračuje, až když analogový výstup `ao1` má hodnotu menší než 5.

Argumenty

```
WaitAO Signal [\LT] | [\GT] Value [\MaxTime] [\ValueAtTimeout]
     [\Visualize] [\Header] [\Message] | [\MsgArray] [\Wrap]
     [\Icon] [\Image] [\VisualizeTime]
```

`Signal`

Datový typ: `signalao`

Jméno analogového výstupního signálu.

`[\LT]`

Less Than

Datový typ: `switch`

Při používání tohoto parametru instrukce `WaitAO` čeká, až hodnota analogového signálu bude menší než hodnota v `Value`.

`[\GT]`

Greater Than

Datový typ: `switch`

Při používání tohoto parametru instrukce `WaitAO` čeká, až hodnota analogového signálu bude větší než hodnota v `Value`.

`Value`

Datový typ: `num`

Požadovaná hodnota signálu.

`[\MaxTime]`

Maximum Time

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Max povolená délka času čekání vyjádřená v sekundách. Jestliže tento čas vyprší před splněním podmínky, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_WAIT_MAXTIME`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

`[\ValueAtTimeout]`

Datový typ: `num`

Jestliže vyprší čas instrukce, aktuální hodnota signálu bude uložena do této proměnné. Proměnná bude nastavena, pouze když systémová proměnná `ERRNO` je nastavena na `ERR_WAIT_MAXTIME`.

`[\Visualize]`

Datový typ: `switch`

Jestliže je zvoleno, aktivuje se vizualizace. Vizualizace se skládá z boxu zpráv s podmínkou, která nebyla splněna, ikony, hlavičky, řádek zprávy a obrázek je zobrazen podle naprogramovaných argumentů.

`[\Header]`

Datový typ: `string`

Text hlavičky je napsán v horní části boxu zpráv. Maximum je 40 znaků. Jestliže není použit žádný argument `\Header`, bude zobrazena výchozí zpráva.

`[\Message]`

Datový typ: `string`

Na displej bude napsána jedna textová řádka. Maximum je 50 znaků.

`[\MsgArray]`

(Message Array)

Datový typ: `string`

Několik textových řádek od pole, které budou napsány na displej. Může být použit pouze jeden z parametrů `\Message` nebo `\MsgArray` ve stejném čase.

Max prostor je 5 řádek s 50 znaky na každé z nich.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 1512*.

Výchozí nastavení, žádná ikona.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.313 WaitAO - Čeká na nastavení hodnoty analogového výstupního signálu
RobotWare - OS
Pokračování

`[\Image]`

Datový typ: `string`

Jméno obrázku, který by měl být použit. Chcete-li poslat své vlastní obrázky, musí být umístěny do adresáře *HOME:* v aktivním systému nebo přímo do aktivního systému.

Doporučuje se umístit soubory do adresáře *HOME:*, aby byly uloženy při provádění zálohování a obnovy.

Vyžaduje se Restart a potom FlexPendant načte obrázky.

Požadavkem na systém je, aby byl použit doplněk RobotWare *FlexPendant Interface*.

Obrázek, který má být zobrazen, může mít šířku 185 a výšku 300. Jestliže je obrázek větší, pouze 185 * 300 pixelů obrázku bude vidět od horní levé části obrázku.

Není možné určovat přesnou hodnotu velikosti obrázků nebo počet obrázků, které mohou být načteny do FlexPendantu. Závisí to na velikosti ostatních souborů načtených do FlexPendant. Vykonávání programu bude pouze pokračovat, jestliže je použit obrázek, který nebyl načten do FlexPendant.

`[\VisualizeTime]`

Datový typ: `num`

Čas čekání před tím, než by se měl objevit box zpráv na FlexPendant. Jestliže používáme argumenty `\VisualizeTime` a `\MaxTime`, čas použitý v argumentu `\MaxTime` musí být delší než čas použitý v argumentu `\VisualizeTime`.

Výchozí čas pro vizualizaci, jestliže nepoužíváme argument `\VisualizeTime` je 5 s. Minimální hodnota 1 s. Maximální hodnota nemá žádný limit. Rozlišení 0.001 s.

Vykonávání programu

Jestliže hodnota signálu je správná, když se vykonává instrukce, program jednoduše pokračuje s následující instrukcí.

Jestliže hodnota signálu je nesprávná, robot vstoupí do stavu čekání a program pokračuje, když se signál změní na správnou hodnotu. Změna je zjištěna přerušením, což dává rychlou odezvu (bez dotazování).

Když robot čeká, čas je dohlížen. Podle výchozího nastavení může robot čekat stále, ale max dobu čekání je možné určit volitelným argumentem `\MaxTime`. Jestliže tento max čas je překročen, je vyvolána chyba.

Jestliže vykonávání programu je zastaveno a později obnoveno, instrukce vyhodnotí aktuální hodnotu signálu. Jakákoliv změna během zastavení programu je odmítnuta.

V ručním režimu, po čekání delším než 3 s se objeví okénko s upozorněním a dotazem, jestli chcete simulovat instrukci. Jestliže nechcete, aby se okénko s upozorněním objevovalo, můžete nastavit systémový parametr *SimulateMenu* na NO, viz *Technická referenční příručka - Systémové parametry*, téma *Controller*, typ *General RAPID*.

Jestliže je použit přepínač `\Visualize`, box zpráv se zobrazí na FlexPendant podle naprogramovaných argumentů. Jestliže není použit žádný argument `\Header`,

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

zobrazí se výchozí text hlavičky. Když je vykonávání instrukce `WaitAO` připraveno, box zpráv bude odstraněn z FlexPendant.

Nový box zpráv na úrovni TRAP přebírá prioritu od boxu zpráv na základní úrovni.

Další příklady

Více příkladů instrukce `WaitAO` je názorně uvedeno dole.

Příklad 1

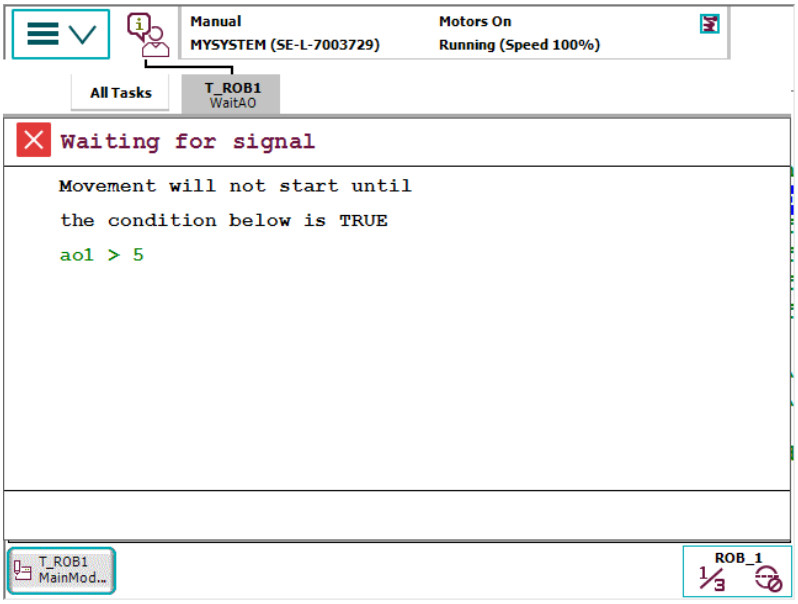
```
VAR num myvalattimeout:=0;
WaitAO ao1, \LT, 5 \MaxTime:=4 \ValueAtTimeout:=myvalattimeout;
ERROR
  IF ERRNO=ERR_WAIT_MAXTIME THEN
    TPWrite "Value of ao1 at timeout:" + ValToStr(myvalattimeout);
    TRYNEXT;
  ELSE
    ! No error recovery handling
  ENDIF
```

Vykonávání programu pokračuje pouze když `ao1` je méně než 5 nebo po vypršení času. Jestliže čas vypršel, hodnota signálu `ao1` při vypršení času může být zapsána bez dalšího čtení signálu.

Příklad 2

```
WaitAO ao1 \GT, 5 \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Jestliže podmínka není splněna, potom bude hlavička a zpráva určená ve volitelných argumentech `\Header` a `\MsgArray` zapsána na displej FlexPendantu společně s podmínkou, které není splněna.



xx1600000151

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.313 WaitAO - Čeká na nastavení hodnoty analogového výstupního signálu

RobotWare - OS

Pokračování

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_AO_LIM, jestliže naprogramovaný argument Value pro určený analogový vstupní signál Signal je mimo limity.

ERR_NO_ALIASIO_DEF jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.

ERR_NORUNUNIT jestliže není žádný kontakt s jednotkou I/O.

ERR_SIG_NOT_VALID jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

ERR_WAIT_MAXTIME, jestliže došlo k vypršení času (parametr \MaxTime) před změnou signálu na správnou hodnotu.

Syntaxe

```
WaitAO
  [ Signal ':=' ] < variable (VAR) of signalao> ´,'
  [ '\' LT] | [ '\' GT] ','
  [ Value ':=' ] < expression (IN) of num>
  ['\'MaxTime ´:='<expression (IN) of num>]
  [ '\'ValueAtTimeout' :=' < variable (VAR) of num >]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Čekání na splnění podmínky	*WaitUntil - Čeká na splnění podmínky na str 965*
Čekání na určený časový úsek	*WaitTime - Čeká danou dobu na str 963*
Čekání na nastavení nebo resetování analogového vstupu	*WaitAI - Čeká na nastavení hodnoty analogového vstupního signálu na str 915*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.314 WaitDI - Čeká na nastavení digitálního vstupního signálu

Použití

`WaitDI` (*Wait Digital Input*) se používá k čekání na nastavení digitálního vstupu.

Základní příklady

Následující příklady názorně ukazují instrukci `WaitDI`:

Příklad 1

```
WaitDI di4, 1;
```

Vykonávání programu pokračuje pouze po nastavení vstupu `di4`.

Příklad 2

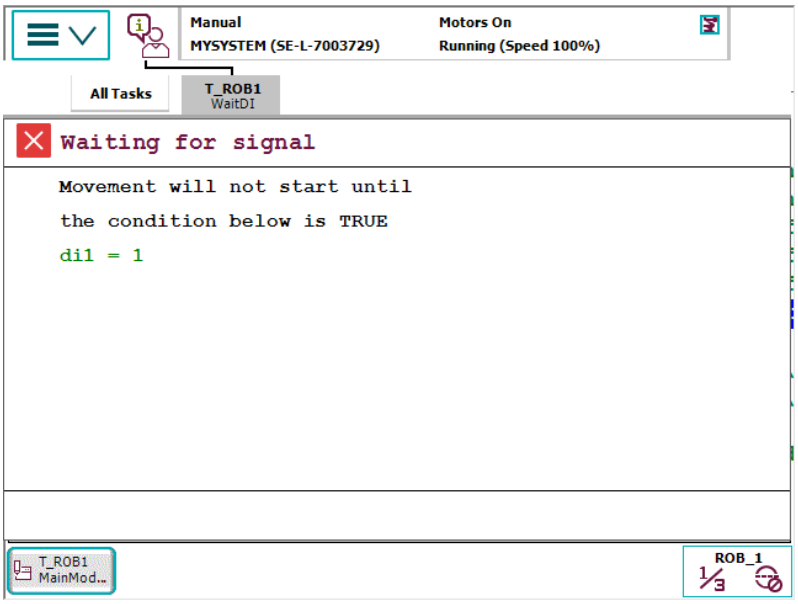
```
WaitDI grip_status, 0;
```

Vykonávání programu pokračuje pouze po resetu vstupu `grip_status`.

Příklad 3

```
WaitDI di1, 1, \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Jestliže podmínka není splněna, potom bude hlavička a zpráva určená ve volitelných argumentech `\Header` a `\MsgArray` zapsána na displej FlexPendantu společně s podmínkou, které není splněna.



xx1600000148

Argumenty

```
WaitDI Signal Value [\MaxTime] [\TimeFlag] [\Visualize] [\Header]
     [\Message] | [\MsgArray] [\Wrap] [\Icon] [\Image]
     [\VisualizeTime]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.314 WaitDI - Čeká na nastavení digitálního vstupního signálu

RobotWare - OS

Pokračování

`Signal`

Datový typ: `signaldi`

Jméno signálu.

`Value`

Datový typ: `dionum`

Požadovaná hodnota signálu.

`[\MaxTime]`

Maximum Time

Datový typ: `num`

Max povolená délka času čekání vyjádřená v sekundách. Jestliže tento čas vyprší před splněním podmínky, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_WAIT_MAXTIME`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

`[\TimeFlag]`

Timeout Flag

Datový typ: `bool`

Výstupní parametr, který obsahuje hodnotu TRUE, jestliže max povolený čas čekání vyprší před splněním podmínky. Jestliže tento parametr je zahrnut v instrukci, potom není považováno za chybu, jestliže max čas vyprší. Tento argument je ignorován, jestliže argument `MaxTime` není zahrnut v instrukci.

`[\Visualize]`

Datový typ: `switch`

Jestliže je zvoleno, aktivuje se vizualizace. Vizualizace se skládá z boxu zpráv s podmínkou, která nebyla splněna, ikony, hlavičky, řádek zprávy a obrázek je zobrazen podle naprogramovaných argumentů.

`[\Header]`

Datový typ: `string`

Text hlavičky je napsán v horní části boxu zpráv. Maximum je 40 znaků. Jestliže není použit žádný argument `\Header`, bude zobrazena výchozí zpráva.

`[\Message]`

Datový typ: `string`

Na displej bude napsána jedna textová řádka. Maximum je 50 znaků.

`[\MsgArray]`

(Message Array)

Datový typ: `string`

Několik textových řádek od pole, které budou napsány na displej. Může být použit pouze jeden z parametrů `\Message` nebo `\MsgArray` ve stejném čase.

Max prostor je 5 řádek s 50 znaky na každé z nich.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 1512*.

Výchozí nastavení, žádná ikona.

`[\Image]`

Datový typ: `string`

Jméno obrázku, který by měl být použit. Chcete-li poslat své vlastní obrázky, musí být umístěny do adresáře *HOME:* v aktivním systému nebo přímo do aktivního systému.

Doporučuje se umístit soubory do adresáře *HOME:*, aby byly uloženy při provádění zálohování a obnovy.

Vyžaduje se Restart a potom FlexPendant načte obrázky.

Požadavkem na systém je, aby byl použit doplněk RobotWare *FlexPendant Interface*.

Obrázek, který má být zobrazen, může mít šířku 185 a výšku 300. Jestliže je obrázek větší, pouze 185 * 300 pixelů obrázku bude vidět od horní levé části obrázku.

Není možné určovat přesnou hodnotu velikosti obrázků nebo počet obrázků, které mohou být načteny do FlexPendantu. Závisí to na velikosti ostatních souborů načtených do FlexPendant. Vykonávání programu bude pouze pokračovat, jestliže je použit obrázek, který nebyl načten do FlexPendant.

`[\VisualizeTime]`

Datový typ: `num`

Čas čekání před tím, než by se měl objevit box zpráv na FlexPendant. Jestliže používáme argumenty `\VisualizeTime` a `\MaxTime`, čas použitý v argumentu `\MaxTime` musí být delší než čas použitý v argumentu `\VisualizeTime`.

Výchozí čas pro vizualizaci, jestliže nepoužíváme argument `\VisualizeTime` je 5 s. Minimální hodnota 1 s. Maximální hodnota nemá žádný limit. Rozlišení 0.001 s.

Vykonávání programu

Jestliže hodnota signálu je správná, když se vykonává instrukce, program jednoduše pokračuje s následující instrukcí.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.314 WaitDI - Čeká na nastavení digitálního vstupního signálu

RobotWare - OS

Pokračování

Jestliže hodnota signálu je nesprávná, robot vstoupí do stavu čekání a program pokračuje, když se signál změní na správnou hodnotu. Změna je zjištěna přerušením, což dává rychlou odezvu (bez dotazování).

Když robot čeká, čas je dohlížen a jestliže je překročena hodnota max času, program bude pokračovat, jestliže je určen `TimeFlag` nebo vyvolá chybu, jestliže tomu tak není. Jestliže je určen `TimeFlag`, bude nastaven na TRUE, jestliže čas byl překročen. Jinak bude nastaven na FALSE.

Jestliže vykonávání programu je zastaveno a později obnoveno, instrukce vyhodnotí aktuální hodnotu signálu. Jakákoliv změna během zastavení programu je odmítnuta.

V ručním režimu, po čekání delším než 3 s se objeví okénko s upozorněním a dotazem, jestli chcete simulovat instrukci. Jestliže nechcete, aby se okénko s upozorněním objevovalo, můžete nastavit systémový parametr *SimulateMenu* na NO, viz *Technická referenční příručka - Systémové parametry*, téma *Controller*, typ *General RAPID*.

Jestliže je použit přepínač `\Visualize`, box zpráv se zobrazí na FlexPendant podle naprogramovaných argumentů. Jestliže není použit žádný argument `\Header`, zobrazí se výchozí text hlavičky. Když je vykonávání instrukce `WaitDI` připraveno, box zpráv bude odstraněn z FlexPendant.

Nový box zpráv na úrovni TRAP přebírá prioritu od boxu zpráv na základní úrovni.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
WaitDI
  [ Signal ':=' ] < variable (VAR) of signaldi>' ,'
  [ Value ':=' ] < expression (IN) of dionum>
  ['\'MaxTime' :='<expression (IN) of num>]
  ['\'TimeFlag':='<variable (VAR) of bool>]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Čekání na splnění podmínky	*WaitUntil - Čeká na splnění podmínky na str 965*
Čekání na určený časový úsek	*WaitTime - Čeká danou dobu na str 963*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.315 WaitDO - Čeká na nastavení digitálního výstupního signálu

Použití

`WaitDO` (*Wait Digital Output*) se používá k čekání na nastavení digitálního výstupu.

Základní příklady

Následující příklady názorně ukazují instrukci `WaitDO`:

Příklad 1

```
WaitDO do4, 1;
```

Vykonávání programu pokračuje pouze po nastavení výstupu `do4`.

Příklad 2

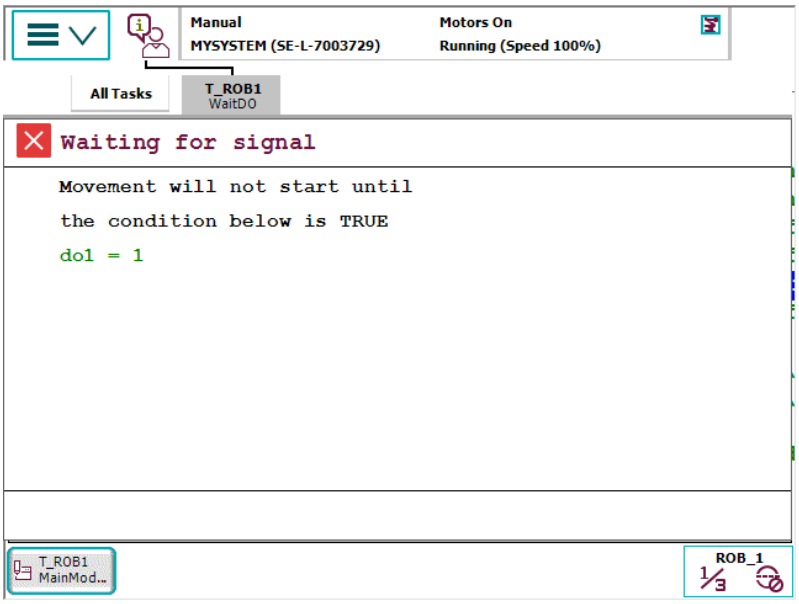
```
WaitDO grip_status, 0;
```

Vykonávání programu pokračuje pouze po nastavení výstupu `grip_status`.

Příklad 3

```
WaitDO do1, 1, \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Jestliže podmínka není splněna, potom bude hlavička a zpráva určená ve volitelných argumentech `\Header` a `\MsgArray` zapsána na displej FlexPendantu společně s podmínkou, které není splněna.



xx1600000149

Argumenty

```
WaitDO Signal Value [\MaxTime] [\TimeFlag] [\Visualize] [\Header]
     [\Message] | [\MsgArray] [\Wrap] [\Icon] [\Image]
     [\VisualizeTime]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Signal`

Datový typ: `signaldo`

Jméno signálu.

`Value`

Datový typ: `dionum`

Požadovaná hodnota signálu.

`[\MaxTime]`

Maximum Time

Datový typ: `num`

Max povolená délka času čekání vyjádřená v sekundách. Jestliže tento čas vyprší před splněním podmínky a argument `TimeFlag` není použit, bude volán chybový handler s chybovým kódem `ERR_WAIT_MAXTIME`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

`[\TimeFlag]`

Timeout Flag

Datový typ: `bool`

Výstupní parametr, který obsahuje hodnotu TRUE, jestliže max povolený čas čekání vyprší před splněním podmínky. Jestliže tento parametr je zahrnut v instrukci, potom není považováno za chybu, jestliže max čas vyprší. Tento argument je ignorován, jestliže argument `MaxTime` není zahrnut v instrukci.

`[\Visualize]`

Datový typ: `switch`

Jestliže je zvoleno, aktivuje se vizualizace. Vizualizace se skládá z boxu zpráv s podmínkou, která nebyla splněna, ikony, hlavičky, řádek zprávy a obrázek je zobrazen podle naprogramovaných argumentů.

`[\Header]`

Datový typ: `string`

Text hlavičky je napsán v horní části boxu zpráv. Maximum je 40 znaků. Jestliže není použit žádný argument `\Header`, bude zobrazena výchozí zpráva.

`[\Message]`

Datový typ: `string`

Na displej bude napsána jedna textová řádka. Maximum je 50 znaků.

`[\MsgArray]`

(Message Array)

Datový typ: `string`

Několik textových řádek od pole, které budou napsány na displej. Může být použit pouze jeden z parametrů `\Message` nebo `\MsgArray` ve stejném čase.

Max prostor je 5 řádek s 50 znaky na každé z nich.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.315 WaitDO - Čeká na nastavení digitálního výstupního signálu
RobotWare - OS
Pokračování

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 1512*.

Výchozí nastavení, žádná ikona.

`[\Image]`

Datový typ: `string`

Jméno obrázku, který by měl být použit. Chcete-li poslat své vlastní obrázky, musí být umístěny do adresáře *HOME:* v aktivním systému nebo přímo do aktivního systému.

Doporučuje se umístit soubory do adresáře *HOME:*, aby byly uloženy při provádění zálohování a obnovy.

Vyžaduje se Restart a potom FlexPendant načte obrázky.

Požadavkem na systém je, aby byl použit doplněk RobotWare *FlexPendant Interface*.

Obrázek, který má být zobrazen, může mít šířku 185 a výšku 300. Jestliže je obrázek větší, pouze 185 * 300 pixelů obrázku bude vidět od horní levé části obrázku.

Není možné určovat přesnou hodnotu velikosti obrázků nebo počet obrázků, které mohou být načteny do FlexPendantu. Závisí to na velikosti ostatních souborů načtených do FlexPendant. Vykonávání programu bude pouze pokračovat, jestliže je použit obrázek, který nebyl načten do FlexPendant.

`[\VisualizeTime]`

Datový typ: `num`

Čas čekání před tím, než by se měl objevit box zpráv na FlexPendant. Jestliže používáme argumenty `\VisualizeTime` a `\MaxTime`, čas použitý v argumentu `\MaxTime` musí být delší než čas použitý v argumentu `\VisualizeTime`.

Výchozí čas pro vizualizaci, jestliže nepoužíváme argument `\VisualizeTime` je 5 s. Minimální hodnota 1 s. Maximální hodnota nemá žádný limit. Rozlišení 0.001 s.

Vykonávání programu

Jestliže hodnota výstupního signálu je správná, když se vykonává instrukce, program jednoduše pokračuje s následující instrukcí.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže hodnota výstupního signálu je nesprávná, robot vstoupí do stavu čekání a program pokračuje, když se signál změní na správnou hodnotu. Změna je zjištěna přerušením, což dává rychlou odezvu (bez dotazování).

Když robot čeká, čas je dohlížen a jestliže je překročena hodnota max času, program bude pokračovat, jestliže je určen `TimeFlag` nebo vyvolá chybu, jestliže tomu tak není. Jestliže je určen `TimeFlag`, bude nastaven na TRUE, jestliže čas byl překročen. Jinak bude nastaven na FALSE.

Jestliže vykonávání programu je zastaveno a později obnoveno, instrukce vyhodnotí aktuální hodnotu signálu. Jakákoliv změna během zastavení programu je odmítnuta.

V ručním režimu, po čekání delším než 3 s se objeví okénko s upozorněním a dotazem, jestli chcete simulovat instrukci. Jestliže nechcete, aby se okénko s upozorněním objevovalo, můžete nastavit systémový parametr *SimulateMenu* na NO, viz *Technická referenční příručka - Systémové parametry*, téma *Controller*, typ *General RAPID*.

Jestliže je použit přepínač `\Visualize`, box zpráv se zobrazí na FlexPendant podle naprogramovaných argumentů. Jestliže není použit žádný argument `\Header`, zobrazí se výchozí text hlavičky. Když je vykonávání instrukce `WaitDO` připraveno, box zpráv bude odstraněn z FlexPendant.

Nový box zpráv na úrovni TRAP přebírá prioritu od boxu zpráv na základní úrovni.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
WaitDO
  [ Signal ´:=´] < variable (VAR) of signaldo >´,´
  [ Value ’:=’ ] < expression (IN) of dionum>
  [’\’MaxTime’ :=’<expression (IN) of num>]
  [’\’TimeFlag’:=’<variable (VAR) of bool>]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.315 WaitDO - Čeká na nastavení digitálního výstupního signálu

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Čekání na splnění podmínky	*WaitUntil - Čeká na splnění podmínky na str 965*
Čekání na určený časový úsek	*WaitTime - Čeká danou dobu na str 963*
Čekání na nastavení nebo resetování vstupu	*WaitDI - Čeká na nastavení digitálního vstupního signálu na str 925*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.316 WaitGI - Čeká na nastavení skupiny digitálních vstupních signálů

Použití

`WaitGI` (*Wait Group digital Input*) se používá pro čekání na nastavení skupiny digitálních vstupních signálů na určené hodnoty.

Základní příklady

Následující příklad názorně ukazuje instrukci `WaitGI`:

Viz také *Další příklady na str 938*.

Příklad 1

```
WaitGI gi4, 5;
```

Vykonávání programu pokračuje, až když vstup `gi4` má hodnotu 5.

Příklad 2

```
WaitGI grip_status, 0;
```

Vykonávání programu pokračuje pouze po resetu vstupu `grip_status`.

Argumenty

```
WaitGI Signal [\NOTEQ] | [\LT] | [\GT] Value | Dvalue [\MaxTime]
     [\ValueAtTimeout] | [\DvalueAtTimeout] [\Visualize] [\Header]
     [\Message] | [\MsgArray] [\Wrap] [\Icon] [\Image]
     [\VisualizeTime]
```

`Signal`

Datový typ: `signalgi`

Jméno digitálního skupinového vstupního signálu.

`[\NOTEQ]`

NOT EQual

Datový typ: `switch`

Při používání tohoto parametru instrukce `WaitGI` čeká, až se hodnota digitálního skupinového signálu oddělí od hodnoty v `Value`.

`[\LT]`

Less Than

Datový typ: `switch`

Při používání tohoto parametru instrukce `WaitGI` čeká, až hodnota digitálního skupinového signálu bude menší než hodnota v `Value`.

`[\GT]`

Greater Than

Datový typ: `switch`

Při používání tohoto parametru instrukce `WaitGI` čeká, až hodnota digitálního skupinového signálu bude větší než hodnota v `Value`.

`Value`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Požadovaná hodnota signálu. Musí to být hodnota celého čísla v rámci pracovního rozsahu použitého digitálního skupinového vstupního signálu. Povolená hodnota je závislá na počtu signálů ve skupině. Max hodnota, kterou je možné použít v argumentu `Value`, je 8388608, a to je hodnota, kterou může mít 23bitový digitální signál jako max hodnotu.

`Dvalue`

Datový typ: `dnum`

Požadovaná hodnota signálu. Musí to být hodnota celého čísla v rámci pracovního rozsahu použitého digitálního skupinového vstupního signálu. Povolená hodnota je závislá na počtu signálů ve skupině. Max množství signálových bitů, kterou může digitální skupinový signál mít, je 32. S proměnnou `dnum` je možné pokrýt rozsah hodnoty 0-4294967295, což je hodnotový rozsah, jaký může 32bitový digitální signál mít.

`[\MaxTime]`

Maximum Time

Datový typ: `num`

Max povolená délka času čekání vyjádřená v sekundách. Jestliže tento čas vyprší před splněním podmínky, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_WAIT_MAXTIME`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

`[\ValueAtTimeout]`

Datový typ: `num`

Jestliže čas instrukce vyprší, aktuální hodnota signálu bude uložena do této proměnné. Proměnná bude nastavena pouze když systémová proměnná `ERRNO` je nastavena na `ERR_WAIT_MAXTIME`. Jestliže je použit argument `Dvalue`, použijte argument `DvalueAtTimeout` pro uložení aktuální hodnoty signálu (důvod: omezení max hodnoty celého čísla pro `num`).

Hodnoty signálu mezi 0 a 8388608 jsou vždy uloženy jako přesné celé číslo.

`[\DvalueAtTimeout]`

Datový typ: `dnum`

Jestliže vyprší čas instrukce, aktuální hodnota signálu bude uložena do této proměnné. Proměnná bude nastavena, pouze když systémová proměnná `ERRNO` je nastavena na `ERR_WAIT_MAXTIME`.

Hodnoty signálu mezi 0 a 4294967295 jsou vždy uloženy jako přesné celé číslo.

`[\Visualize]`

Datový typ: `switch`

Jestliže je zvoleno, aktivuje se vizualizace. Vizualizace se skládá z boxu zpráv s podmínkou, která nebyla splněna, ikony, hlavičky, řádek zprávy a obrázek je zobrazen podle naprogramovaných argumentů.

`[\Header]`

Datový typ: `string`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Text hlavičky je napsán v horní části boxu zpráv. Maximum je 40 znaků. Jestliže není použit žádný argument `\Header`, bude zobrazena výchozí zpráva.

`[\Message]`

Datový typ: `string`

Na displej bude napsána jedna textová řádka. Maximum je 50 znaků.

`[\MsgArray]`

(Message Array)

Datový typ: `string`

Několik textových řádek od pole, které budou napsány na displej. Může být použit pouze jeden z parametrů `\Message` nebo `\MsgArray` ve stejném čase.

Max prostor je 5 řádek s 50 znaky na každé z nich.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 1512*.

Výchozí nastavení, žádná ikona.

`[\Image]`

Datový typ: `string`

Jméno obrázku, který by měl být použit. Chcete-li poslat své vlastní obrázky, musí být umístěny do adresáře *HOME:* v aktivním systému nebo přímo do aktivního systému.

Doporučuje se umístit soubory do adresáře *HOME:*, aby byly uloženy při provádění zálohování a obnovy.

Vyžaduje se Restart a potom FlexPendant načte obrázky.

Požadavkem na systém je, aby byl použit doplněk RobotWare *FlexPendant Interface*.

Obrázek, který má být zobrazen, může mít šířku 185 a výšku 300. Jestliže je obrázek větší, pouze 185 * 300 pixelů obrázku bude vidět od horní levé části obrázku.

Není možné určovat přesnou hodnotu velikosti obrázků nebo počet obrázků, které mohou být načteny do FlexPendantu. Závisí to na velikosti ostatních souborů načtených do FlexPendant. Vykonávání programu bude pouze pokračovat, jestliže je použit obrázek, který nebyl načten do FlexPendant.

`[\VisualizeTime]`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.316 WaitGI - Čeká na nastavení skupiny digitálních vstupních signálů
RobotWare - OS
Pokračování

Čas čekání před tím, než by se měl objevit box zpráv na FlexPendant. Jestliže používáme argumenty `\VisualizeTime` a `\MaxTime`, čas použitý v argumentu `\MaxTime` musí být delší než čas použitý v argumentu `\VisualizeTime`.

Výchozí čas pro vizualizaci, jestliže nepoužíváme argument `\VisualizeTime` je 5 s. Minimální hodnota 1 s. Maximální hodnota nemá žádný limit. Rozlišení 0.001 s.

Vykonávání programu

Jestliže hodnota signálu je správná, když se vykonává instrukce, program jednoduše pokračuje s následující instrukcí.

Jestliže hodnota signálu je nesprávná, robot vstoupí do stavu čekání a program pokračuje, když se signál změní na správnou hodnotu. Změna je zjištěna přerušením, což dává rychlou odezvu (bez dotazování).

Když robot čeká, čas je dohlížen. Podle výchozího nastavení může robot čekat stále, ale max dobu čekání je možné určit volitelným argumentem `\MaxTime`. Jestliže tento max čas je překročen, je vyvolána chyba.

Jestliže vykonávání programu je zastaveno a později obnoveno, instrukce vyhodnotí aktuální hodnotu signálu. Jakákoliv změna během zastavení programu je odmítnuta.

V ručním režimu, po čekání delším než 3 s se objeví okénko s upozorněním a dotazem, jestli chcete simulovat instrukci. Jestliže nechcete, aby se okénko s upozorněním objevovalo, můžete nastavit systémový parametr *SimulateMenu* na NO, viz *Technická referenční příručka - Systémové parametry*, téma *Controller*, typ *General RAPID*.

Jestliže je použit přepínač `\Visualize`, box zpráv se zobrazí na FlexPendant podle naprogramovaných argumentů. Jestliže není použit žádný argument `\Header`, zobrazí se výchozí text hlavičky. Když je vykonávání instrukce `WaitGI` připraveno, box zpráv bude odstraněn z FlexPendant.

Nový box zpráv na úrovni TRAP přebírá prioritu od boxu zpráv na základní úrovni.

Další příklady

Více příkladů instrukce `WaitGI` je názorně uvedeno dole.

Příklad 1

```
WaitGI gi1,\NOTEQ,0;
```

Vykonávání programu pokračuje, až když se `gi1` liší od hodnoty 0.

Příklad 2

```
WaitGI gi1,\LT,1;
```

Vykonávání programu pokračuje, až když je `gi1` méně než 1.

Příklad 3

```
WaitGI gi1,\GT,0;
```

Vykonávání programu pokračuje, až když je `gi1` větší než 0.

Příklad 4

```
VAR num myvalattimeout:=0;
WaitGI gi1, 5 \MaxTime:=4 \ValueAtTimeout:=myvalattimeout;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

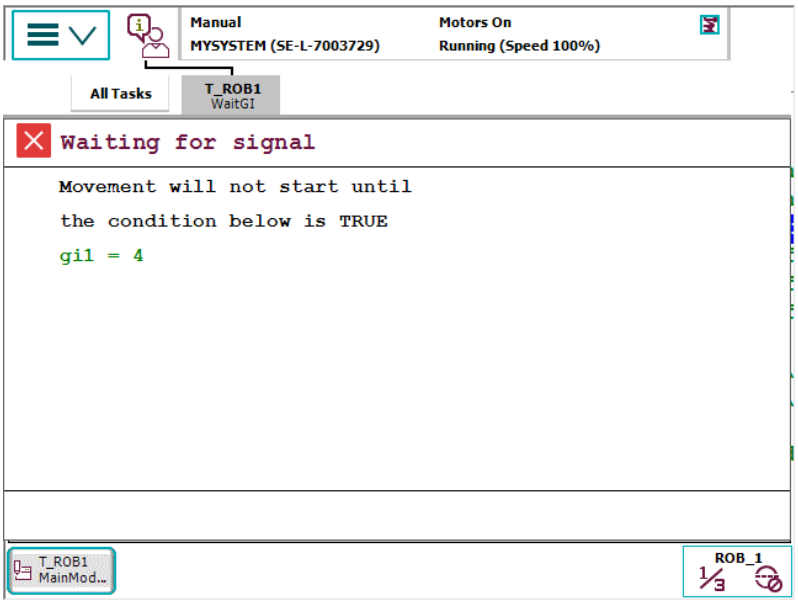
```
ERROR
  IF ERRNO=ERR_WAIT_MAXTIME THEN
    TPWrite "Value of gi1 at timeout:" + ValToStr(myvalattimeout);
    TRYNEXT;
  ELSE
    ! No error recovery handling
  ENDIF
```

Vykonávání programu pokračuje pouze když `gi1` se rovná 5 nebo po vypršení času. Jestliže čas vypršel, hodnota signálu `gi1` při vypršení času může být zapsána bez dalšího čtení signálu.

Příklad 5

```
WaitGI gi1, 4, \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Jestliže podmínka není splněna, potom bude hlavička a zpráva určená ve volitelných argumentech `\Header` a `\MsgArray` zapsána na displej FlexPendantu společně s podmínkou, které není splněna.



xx1600000152

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_GO_LIM`, jestliže naprogramovaný argument `Value` nebo `Dvalue` pro určený digitální skupinový vstupní signál `Signal` je mimo limity.

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.316 WaitGI - Čeká na nastavení skupiny digitálních vstupních signálů

RobotWare - OS

Pokračování

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

`ERR_WAIT_MAXTIME`, jestliže došlo k vypršení času (parametr `\MaxTime`) před změnou signálu na správnou hodnotu.

Syntaxe

```
WaitGI
  [ Signal ':=' ] < variable (VAR) of signalgi> ´,´
  ['\' NOTEQ] | [ '\' LT] | [ '\' GT] ','
  [ Value ':=' ] < expression (IN) of num>
  | [ Dvalue' :=' ] < expression (IN) of dnum>
  ['\'MaxTime ´:='<expression (IN) of num>]
  [ '\'ValueAtTimeout' :=' < variable (VAR) of num > ]
  | [ '\'DvalueAtTimeout' :=' < variable (VAR) of dnum > ]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Čekání na splnění podmínky	*WaitUntil - Čeká na splnění podmínky na str 965*
Čekání na určený časový úsek	*WaitTime - Čeká danou dobu na str 963*
Čekejte do nastavení/resetování skupiny digitálních výstupních signálů	*WaitGO - Čeká do nastavení skupiny digitálních výstupních signálů na str 941*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.317 WaitGO - Čeká do nastavení skupiny digitálních výstupních signálů

Použití

`WaitGO` (*Wait Group digital Output*) se používá pro čekání na nastavení skupiny digitálních výstupních signálů na určenou hodnotu.

Základní příklady

Následující příklady názorně ukazují instrukci `WaitGO`:

Viz také *Další příklady na str 944*.

Příklad 1

```
WaitGO go4, 5;
```

Vykonávání programu pokračuje, až když výstup `go4` má hodnotu 5.

Příklad 2

```
WaitGO grip_status, 0;
```

Vykonávání programu pokračuje až po resetování výstupu `grip_status`.

Argumenty

```
WaitGO Signal [\NOTEQ] | [\LT] | [\GT] Value | Dvalue [\MaxTime]
     [\ValueAtTimeout] | [\DvalueAtTimeout] [\Visualize] [\Header]
     [\Message] | [\MsgArray] [\Wrap] [\Icon] [\Image]
     [\VisualizeTime]
```

`Signal`

Datový typ: `signalgo`

Jméno digitálního skupinového výstupního signálu.

`[\NOTEQ]`

NOT EQual

Datový typ: `switch`

Při používání tohoto parametru instrukce `WaitGO` čeká, až se hodnota digitálního skupinového signálu oddělí od hodnoty v `Value`.

`[\LT]`

Less Than

Datový typ: `switch`

Při používání tohoto parametru instrukce `WaitGO` čeká, až hodnota digitálního skupinového signálu bude menší než hodnota v `Value`.

`[\GT]`

Greater Than

Datový typ: `switch`

Při používání tohoto parametru instrukce `WaitGO` čeká, až hodnota digitálního skupinového signálu bude větší než hodnota v `Value`.

`Value`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.317 WaitGO - Čeká do nastavení skupiny digitálních výstupních signálů
RobotWare - OS
Pokračování

Požadovaná hodnota signálu. Musí to být hodnota celého čísla v rámci pracovního rozsahu použitého digitálního skupinového výstupního signálu. Povolená hodnota je závislá na počtu signálů ve skupině. Max hodnota, kterou je možné použít v argumentu `Value`, je 8388608, a to je hodnota, kterou může mít 23bitový digitální signál jako max hodnotu.

`Dvalue`

Datový typ: `dnum`

Požadovaná hodnota signálu. Musí to být hodnota celého čísla v rámci pracovního rozsahu použitého digitálního skupinového výstupního signálu. Povolená hodnota je závislá na počtu signálů ve skupině. Max množství signálových bitů, kterou může digitální skupinový signál mít, je 32. S proměnnou `dnum` je možné pokrýt rozsah hodnoty 0-4294967295, což je hodnotový rozsah, jaký může 32bitový digitální signál mít.

`[\MaxTime]`

Maximum Time

Datový typ: `num`

Max povolená délka času čekání vyjádřená v sekundách. Jestliže tento čas vyprší před splněním podmínky, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_WAIT_MAXTIME`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

`[\ValueAtTimeout]`

Datový typ: `num`

Jestliže čas instrukce vyprší, aktuální hodnota signálu bude uložena do této proměnné. Proměnná bude nastavena pouze když systémová proměnná `ERRNO` je nastavena na `ERR_WAIT_MAXTIME`. Jestliže je použit argument `Dvalue`, použijte argument `DvalueAtTimeout` pro uložení aktuální hodnoty signálu (důvod: omezení max hodnoty celého čísla pro `num`).

Hodnoty signálu mezi 0 a 8388608 jsou vždy uloženy jako přesné celé číslo.

`[\DvalueAtTimeout]`

Datový typ: `dnum`

Jestliže vyprší čas instrukce, aktuální hodnota signálu bude uložena do této proměnné. Proměnná bude nastavena, pouze když systémová proměnná `ERRNO` je nastavena na `ERR_WAIT_MAXTIME`.

Hodnoty signálu mezi 0 a 4294967295 jsou vždy uloženy jako přesné celé číslo.

`[\Visualize]`

Datový typ: `switch`

Jestliže je zvoleno, aktivuje se vizualizace. Vizualizace se skládá z boxu zpráv s podmínkou, která nebyla splněna, ikony, hlavičky, řádek zprávy a obrázek je zobrazen podle naprogramovaných argumentů.

`[\Header]`

Datový typ: `string`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Text hlavičky je napsán v horní části boxu zpráv. Maximum je 40 znaků. Jestliže není použit žádný argument `\Header`, bude zobrazena výchozí zpráva.

`[\Message]`

Datový typ: `string`

Na displej bude napsána jedna textová řádka. Maximum je 50 znaků.

`[\MsgArray]`

(Message Array)

Datový typ: `string`

Několik textových řádek od pole, které budou napsány na displej. Může být použit pouze jeden z parametrů `\Message` nebo `\MsgArray` ve stejném čase.

Max prostor je 5 řádek s 50 znaky na každé z nich.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 1512*.

Výchozí nastavení, žádná ikona.

`[\Image]`

Datový typ: `string`

Jméno obrázku, který by měl být použit. Chcete-li poslat své vlastní obrázky, musí být umístěny do adresáře *HOME:* v aktivním systému nebo přímo do aktivního systému.

Doporučuje se umístit soubory do adresáře *HOME:*, aby byly uloženy při provádění zálohování a obnovy.

Vyžaduje se Restart a potom FlexPendant načte obrázky.

Požadavkem na systém je, aby byl použit doplněk RobotWare *FlexPendant Interface*.

Obrázek, který má být zobrazen, může mít šířku 185 a výšku 300. Jestliže je obrázek větší, pouze 185 * 300 pixelů obrázku bude vidět od horní levé části obrázku.

Není možné určovat přesnou hodnotu velikosti obrázků nebo počet obrázků, které mohou být načteny do FlexPendantu. Závisí to na velikosti ostatních souborů načtených do FlexPendant. Vykonávání programu bude pouze pokračovat, jestliže je použit obrázek, který nebyl načten do FlexPendant.

`[\VisualizeTime]`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.317 WaitGO - Čeká do nastavení skupiny digitálních výstupních signálů
RobotWare - OS
Pokračování

Čas čekání před tím, než by se měl objevit box zpráv na FlexPendant. Jestliže používáme argumenty `\VisualizeTime` a `\MaxTime`, čas použitý v argumentu `\MaxTime` musí být delší než čas použitý v argumentu `\VisualizeTime`.

Výchozí čas pro vizualizaci, jestliže nepoužíváme argument `\VisualizeTime` je 5 s. Minimální hodnota 1 s. Maximální hodnota nemá žádný limit. Rozlišení 0.001 s.

Vykonávání programu

Jestliže hodnota signálu je správná, když se vykonává instrukce, program jednoduše pokračuje s následující instrukcí.

Jestliže hodnota signálu je nesprávná, robot vstoupí do stavu čekání a program pokračuje, když se signál změní na správnou hodnotu. Změna je zjištěna přerušením, což dává rychlou odezvu (bez dotazování).

Když robot čeká, čas je dohlížen. Podle výchozího nastavení může robot čekat stále, ale max dobu čekání je možné určit volitelným argumentem `\MaxTime`. Jestliže tento max čas je překročen, je vyvolána chyba.

Jestliže vykonávání programu je zastaveno a později obnoveno, instrukce vyhodnotí aktuální hodnotu signálu. Jakákoliv změna během zastavení programu je odmítnuta.

V ručním režimu, po čekání delším než 3 s se objeví okénko s upozorněním a dotazem, jestli chcete simulovat instrukci. Jestliže nechcete, aby se okénko s upozorněním objevovalo, můžete nastavit systémový parametr *SimulateMenu* na NO, viz *Technická referenční příručka - Systémové parametry*, téma *Controller*, typ *General RAPID*.

Jestliže je použit přepínač `\Visualize`, box zpráv se zobrazí na FlexPendant podle naprogramovaných argumentů. Jestliže není použit žádný argument `\Header`, zobrazí se výchozí text hlavičky. Když je vykonávání instrukce `WaitGO` připraveno, box zpráv bude odstraněn z FlexPendant.

Nový box zpráv na úrovni TRAP přebírá prioritu od boxu zpráv na základní úrovni.

Další příklady

Více příkladů instrukce `WaitGO` je názorně uvedeno dole.

Příklad 1

```
WaitGO go1,\NOTEQ,0;
```

Vykonávání programu pokračuje, až když se `go1` liší od hodnoty 0.

Příklad 2

```
WaitGO go1,\LT,1;
```

Vykonávání programu pokračuje, až když je `go1` méně než 1.

Příklad 3

```
WaitGO go1,\GT,0;
```

Vykonávání programu pokračuje, až když je `go1` větší než 0.

Příklad 4

```
VAR num myvalattimeout:=0;
WaitGO go1, 5 \MaxTime:=4 \ValueAtTimeout:=myvalattimeout;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

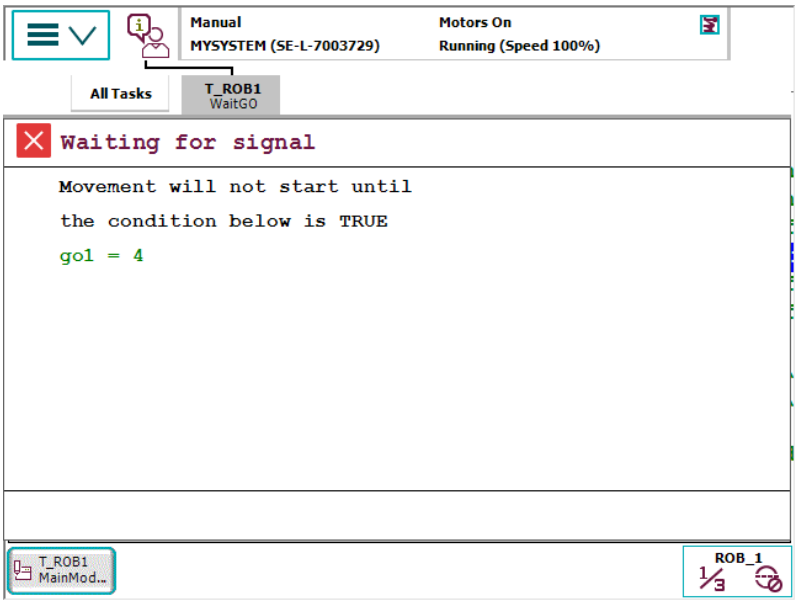
```
ERROR
  IF ERRNO=ERR_WAIT_MAXTIME THEN
    TPWrite "Value of go1 at timeout:" + ValToStr(myvalattimeout);
    TRYNEXT;
  ELSE
    ! No error recovery handling
  ENDIF
```

Vykonávání programu pokračuje pouze když `go1` se rovná 5 nebo po vypršení času. Jestliže čas vypršel, hodnota signálu `go1` při vypršení času může být zapsána bez dalšího čtení signálu.

Příklad 5

```
WaitGO go1, 4, \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Jestliže podmínka není splněna, potom bude hlavička a zpráva určená ve volitelných argumentech `\Header` a `\MsgArray` zapsána na displej FlexPendantu společně s podmínkou, které není splněna.



xx1600000153

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_GO_LIM`, jestliže naprogramovaný argument `Value` nebo `Dvalue` pro určený digitální skupinový výstupní signál `Signal` je mimo limity.

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.317 WaitGO - Čeká do nastavení skupiny digitálních výstupních signálů

RobotWare - OS

Pokračování

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

`ERR_WAIT_MAXTIME`, jestliže došlo k vypršení času (parametr `\MaxTime`) před změnou signálu na správnou hodnotu.

Syntaxe

```
WaitGO
  [ Signal ':=' ] < variable (VAR) of signalgo> ','
  ['\' NOTEQ] | [ '\' LT] | [ '\' GT] ','
  [ Value ':=' ] < expression (IN) of num>
  | [ Dvalue ':=' ] < expression (IN) of dnum>
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' ValueAtTimeout ':=' < variable (VAR) of num > ]
  | [ '\' DvalueAtTimeout ':=' < variable (VAR) of dnum > ]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Čekání na splnění podmínky	*WaitUntil - Čeká na splnění podmínky na str 965*
Čekání na určený časový úsek	*WaitTime - Čeká danou dobu na str 963*
Čekání do nastavení/resetování skupiny digitálních vstupních signálů	*WaitGI - Čeká na nastavení skupiny digitálních vstupních signálů na str 935*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.318 WaitLoad - Připojit načtený modul k úloze

Použití

`WaitLoad` se používá pro spojení s modulem, který je načten s instrukcí `StartLoad` do programové úlohy.

Načtený programový modul bude přidán k již existujícím modulům v paměti programu.

Modul, který je načten s `StartLoad`, musí být připojen k programové úloze s instrukcí `WaitLoad` předtím, než je možné používat jeho symboly nebo rutiny.

`WaitLoad` může také stáhnout programový modul, jestliže jsou použity volitelné přepínače. Tím se minimalizuje počet odkazů (1 namísto 2).

`WaitLoad` může také kontrolovat nevyřešené reference, jestliže je použit volitelný přepínač `\CheckRef`.

Základní příklady

Následující příklad názorně ukazuje instrukci `WaitLoad`:

Viz také *Další příklady na str 948*.

Příklad 1

```
VAR loadsession load1;
...
StartLoad "HOME:/PART_A.MOD", load1;
MoveL p10, v1000, z50, tool1 \WObj:=wobj1;
MoveL p20, v1000, z50, tool1 \WObj:=wobj1;
MoveL p30, v1000, z50, tool1 \WObj:=wobj1;
MoveL p40, v1000, z50, tool1 \WObj:=wobj1;
WaitLoad load1;
%"routine_x"%;
UnLoad "HOME:/PART_A.MOD";
```

Načtěte programový modul `PART_A.MOD` z `HOME:` do paměti programu. Současně posuňte robot. Potom připojte nový programový modul k programové úloze a zavolejte rutinu `routine_x` v modulu `PART_A`.

Argumenty

```
WaitLoad [\UnloadPath] [\UnloadFile] LoadNo [\CheckRef]
```

`[\UnloadPath]`

Datový typ: `string`

Cesta souboru a jméno souboru k souboru, který bude stažen z paměti programu. Jméno souboru by mělo být vyřazeno, když se používá argument `\UnloadFile`.

`[\UnloadFile]`

Datový typ: `string`

Když je jméno souboru vyřazeno v argumentu `\UnloadPath`, potom musí být definováno s tímto argumentem.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.318 WaitLoad - Připojit načtený modul k úloze

RobotWare - OS

Pokračování

`LoadNo`

Datový typ: `loadsession`

Toto je reference k načítací akci, vytvořené instrukcí `StartLoad`, která je nutná kvůli připojení načteného programového modulu k programové úloze.

`[\CheckRef]`

Datový typ: `switch`

Po načtení modulu zkontrolujte nevyřešené reference v programové úloze. Jinak nebude žádná kontrola nevyřešených referencí provedena.

Vykonávání programu

Instrukce `WaitLoad` bude nejprve čekat na dokončení načítání, pokud nebylo dosud dokončeno, a potom bude modul propojen a inicializován. Iniciace načteného modulu nastavuje všechny proměnné na úrovni modulu na jejich počáteční hodnoty.

Nevyřešené reference budou vždy akceptovány pro operace načítání `StartLoad - WaitLoad`, jestliže není použit parametr `\CheckRef`, ale bude to chyba při běhu při vykonávání nevyřešené reference.

Systém zahájí operaci stahování, jestliže je určena. Jestliže stahování modulu selže, nový modul nebude načten.

Po jakékoliv chybě z operace načítání včetně nevyřešených referencí při použití přepínače `\CheckRef` nebude už načtený modul k dispozici v paměti programu.

Aby bylo možné získat dobrou programovou strukturu, která je snadná na porozumění i udržování, veškeré načítání a stahování programových modulů by se mělo provádět od hlavního modulu, který je vždy přítomen v paměti programu během vykonávání.

Pro načítání programu, který obsahuje main proceduru k main programu (s další procedurou `main`), viz instrukce `Load`.

Další příklady

Více příkladů instrukce `WaitLoad` je názorně uvedeno dole.

Příklad 1

```
StartLoad "HOME:/DOORDIR/DOOR2.MOD", load1;
...
WaitLoad \UnloadPath:="HOME:/DOORDIR/DOOR1.MOD", load1;
```

Načtěte programový modul `DOOR2.MOD` z`HOME:` v adresáři `DOORDIR` do paměti programu a připojte nový modul k úloze. Programový modul `DOOR1.MOD` bude stažen z paměti programu.

Příklad 2

```
StartLoad "HOME:" \File:="DOORDIR/DOOR2.MOD", load1;
! The robot can do some other work
WaitLoad \UnloadPath:="HOME:" \File:= "DOORDIR/DOOR1.MOD", load1;
```

Je to stejné jako instrukce dole, ale robot může dělat nějakou jinou práci během doby načítání a také dělat ji rychleji (pouze jeden odkaz namísto dvou odkazů dole).

```
Load "HOME:" \File:="DOORDIR/DOOR2.MOD";
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
UnLoad "HOME:" \File:="DOORDIR/DOOR1.MOD";
```

Řešení chyb

Jestliže soubor určený v instrukci `StartLoad` není možné nalézt, systémová proměnná `ERRNO` je nastavena na `ERR_FILNOTFND` při vykonávání `WaitLoad`.

Jestliže se vyskytne jiný typ problémů při čtení souboru k načtení, potom bude systémová proměnná `ERRNO` nastavena na `ERR_IOERROR`.

Jestliže argument `LoadNo` odkazuje na neznámou načítací akci, potom bude systémová proměnná `ERRNO` nastavena na `ERR_UNKPROC`.

Jestliže modul nemůže být načten, protože paměť programu je plná, potom bude systémová proměnná `ERRNO` nastavena na `ERR_PRGMEMFULL`.

Jestliže modul je již načten do paměti programu, potom bude systémová proměnná `ERRNO` nastavena na `ERR_LOADED`.

Jestliže načtený modul obsahuje chyby syntaxe, systémová proměnná `ERRNO` je nastavena na `ERR_SYNTAX`.

Jestliže výsledkem načteného modulu jsou fatální odkazové chyby, systémová proměnná `ERRNO` je nastavena na `ERR_LINKREF`.

Jestliže se používá `WaitLoad` s přepínačem `CheckRef` pro kontrolu referenčních chyb, a paměť programu obsahuje nevyřešené reference, systémová proměnná `ERRNO` je nastavena na `ERR_LINKREF`.

Následující chyby mohou vzniknout, pouze když argument `\UnloadPath` je použit v instrukci `WaitLoad`:

- Jestliže modul, určený v argumentu `\UnloadPath` nemůže být načten kvůli probíhajícímu vykonávání v rámci modulu, potom je systémová proměnná ERRNO nastavena na `ERR_UNLOAD`.
- Jestliže modul, určený v argumentu `\UnloadPath` nemůže být načten, protože programový modul není načten s `Load` nebo `StartLoad-WaitLoad` z programu RAPID, potom je systémová proměnná `ERRNO` také nastavena na `ERR_UNLOAD`.

Tyto chyby mohou být potom ošetřeny v chybovém handleru `ERROR`. Jestliže se objevuje některá z těchto chyb, aktuální modul bude stažen a nebude k dispozici v handleru `ERROR`.

POZNÁMKA

RETRY se nemůže používat pro obnovu po chybě u jakýchkoliv chyb z `WaitLoad`.

Omezení

Není možné měnit aktuální hodnotu některé proměnné PERS načtením stejného modulu s novou init hodnotou pro aktuální proměnnou PERS.

Příklad:

- Soubor `my_module.mod` s deklarací `PERS num my_pers:=1;` je načten do systému.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.318 WaitLoad - Připojit načtený modul k úloze

RobotWare - OS

Pokračování

- Soubor `my_module.mod` je editován na disku s novou perzistentní hodnotou např. `PERS num my_pers:=3;`
- Kód dole je vykonán.
- Po novém načtení `my_module.mod` je hodnota `my_pers` stále `1` namísto `3`.

```
StartLoad \Dynamic, "HOME:/my_module.mod", load1;
...
WaitLoad \UnLoadPath:="HOME:/my_module.mod", load1;
```

Toto omezení je následkem vlastností proměnné `PERS`. Aktuální hodnota proměnné `PERS` nebude změněna nově načtenou init hodnotou `PERS`, jestliže proměnná `PERS` se jakkoliv používá v době načítání.

Shora uvedené problémy nevzniknou, jestliže je namísto toho vykonán následující kód:

```
UnLoad "HOME:/my_module.mod";
StartLoad \Dynamic, "HOME:/my_module.mod", load1;
...
WaitLoad load1;
```

Další možností je použít CONST pro init hodnotu a provést následující přidělení na začátku vykonávání nového modulu: `my_pers := my_const;`

Syntaxe

WaitLoad

```
[ '\' UnloadPath ':=' <expression (IN) of string> ',']
[ '\' UnloadFile ':=' <expression (IN) of string> ',']
[ LoadNo ':=' ] <variable (VAR) of loadsession>
[ '\' CheckRef ] ';'
```

Související informace

Pro informace o	Viz
Načíst programový modul během provádění	*StartLoad - Načíst programový modul během vykonávání na str 703*
Načíst akci	*loadsession - Načtení programu na str 1530*
Načíst programový modul	*Load - Načíst programový modul během provádění na str 328*
Zrušit načtení programového modulu	*UnLoad - Stáhnout programový modul během provádění na str 905*
Zrušit načítání programového modulu	*CancelLoad - Zrušit načítání modulu na str 64*
Zkontrolovat reference programu	*CheckProgRef - Zkontrolovat reference programu na str 103*
Volání procedury s opožděnou vazbou	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.319 WaitRob - Čekat na stop bod nebo nulovou rychlost

Použití

`WaitRob` (*Wait Robot*) Čeká, až robot a externí osy dosáhnou stop bodu nebo mají nulovou rychlost.

Základní příklady

Následující příklad názorně ukazuje instrukci `WaitRob`:

Viz také *Další příklady na str 951*.

Příklad 1

```
WaitRob \InPos;
```

Vykonávání programu čeká, až robot a externí osy dosáhnou stop bodu.

Argumenty

```
WaitRob [\InPos] | [\ZeroSpeed]
```

`[\InPos]`

In Position

Datový typ: `switch`

Jestliže je použit tento argument, robot a externí osy musí dosáhnout stop bodu (`ToPoint` aktuální pohybové instrukce), potom může vykonávání pokračovat.

`[\ZeroSpeed]`

Zero Speed

Datový typ: `switch`

Jestliže je použit tento argument, robot a externí osy musí mít nulovou rychlost, potom může vykonávání pokračovat.

Jestliže není zapsán žádný z argumentů `\InPos` nebo `\ZeroSpeed`, zobrazí se chybová zpráva.

Další příklady

Více příkladů jak používat instrukci `WaitRob` je názorně uvedeno dole.

Příklad 1

```
PROC stop_event()
  WaitRob \ZeroSpeed;
  SetDO rob_moving, 0;
ENDPROC
```

Příklad ukazuje událostní rutinu, která vykonává při zastavení programu. Digitální výstupní signál `rob_moving` je 1, dokud se robot pohybuje, a je nastaven na 0, když robot a externí osy zastaví pohyb po zastavení programu.

Syntaxe

```
WaitRob
  [ '\' InPos ] | [ '\' ZeroSpeed ] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.319 WaitRob - Čekat na stop bod nebo nulovou rychlost

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O*
Ostatní polohovací instrukce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Definice dat stop bodu	*stoppointdata - Data stop bodu na str 1590*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.320 WaitSensor - Čekat na připojení na senzor

Použití

`WaitSensor` **Připojuje k objektu v okně startu na mechanické jednotce senzoru.**

Základní příklady

Základní příklady instrukce `WaitSensor` jsou názorně uvedeny dole.

Viz také *Další příklady na str 954*.

Příklad 1

```
WaitSensor Ssync1;
```

Program se připojuje k prvnímu objektu ve frontě objektů, tj. v rámci spouštěcího okna na senzoru. Jestliže ve spouštěcím okně není žádný objekt, vykonávání se zastaví a čeká na objekt.

Argumenty

```
WaitSensor MechUnit [\RelDist ][\PredTime][\MaxTime][\TimeFlag]
```

`MechUnit`

Mechanická jednotka

Datový typ: `mecunit`

Pohybující se mechanická jednotka, ke které se vztahuje pozice robotu v instrukci.

`[\RelDist]`

Relativní vzdálenost

Datový typ: `num`

Čeká na objekt před vstoupením do spouštěcího okna a přechází za vzdálenost určenou v argumentu. Jestliže pracovní objekt je již připojen, vykonávání se zastaví, dokud objekt nepřejde danou vzdálenost. Jestliže objekt už přešel za relativní vzdálenost, vykonávání pokračuje.

`[\PredTime]`

Doba předpovědi

Datový typ: `num`

Čeká na objekt před vstoupením do spouštěcího okna a přechází za vzdálenost určenou argumentem. Jestliže pracovní objekt je již připojen, vykonávání se zastaví, dokud objekt nepřejde danou vzdálenost. Jestliže objekt už překročil dobu předpovědi, vykonávání pokračuje.

`[\MaxTime]`

Maximální čas

Datový typ: `num`

Max povolená délka doby čekání vyjádřená v sekundách. Jestliže tento čas vyprší před připojením senzoru nebo dosažením `\RelDist`, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_WAIT_MAXTIME`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.320 WaitSensor - Čekat na připojení na senzor

Machine Synchronization
Pokračování

`[\TimeFlag]`

Příznak vypršení času

Datový typ: `bool`

Výstupní parametr, který obsahuje hodnotu `TRUE`, jestliže max povolený čas čekání vyprší před připojením senzoru nebo dosažením `\RelDist`. Jestliže tento parametr je zahrnut v instrukci, potom není považováno za chybu, jestliže max čas vyprší.

Tento argument je ignorován, jestliže argument `MaxTime` není zahrnut do instrukce.

Vykonávání programu

Jestliže ve spouštěcím okně není žádný objekt, vykonávání programu se zastaví. Jestliže je objekt přítomný, potom je objekt připojen k senzoru a vykonávání pokračuje.

Jestliže druhá instrukce `WaitSensor` je vydána při připojení, je vrácena chyba, pokud není použit volitelný argument `\RelDist`.

Další příklady

Více příkladů instrukce je názorně uvedeno dole.

Příklad 1

```
WaitSensor Ssync1\RelDist:=500.0;
```

Jestliže není připojeno, čekejte na objekt, až vstoupí do spouštěcího okna, a potom čekejte na objekt, až přejde bod 500 mm na senzoru.

Jestliže jste již připojeni k objektu, potom čekejte, až objekt přejde 500 mm.

Příklad 2

```
WaitSensor Ssync1\RelDist:=0.0;
```

Jestliže nejste připojeni, potom čekejte na objekt ve spouštěcím okně.

Jestliže jste již připojeni, potom pokračujte s vykonáváním, protože objekt již přešel přes 0,0 mm.

Příklad 3

```
WaitSensor Ssync1;
WaitSensor Ssync1\RelDist:=0.0;
```

První `WaitSensor` se připojuje k objektu ve spouštěcím okně. Druhý `WaitSensor` se bude okamžitě vracet, jestliže objekt je stále připojen, ale bude čekat na další objekt, jestliže předchozí objekt se posunul za max vzdálenost nebo byl vyhozen.

Příklad 4

```
WaitSensor Ssync1\RelDist:=0.5\PredTime:=0.1;
```

`WaitSensor` se bude okamžitě vracet, jestliže objekt přešel 0,5 metru, ale jinak bude čekat, až objekt dosáhne *=RelDist - C1speed * PredTime*. Cílem je předvídat prodlevy před spuštěním nové pohybové instrukce.

Příklad 5

```
WaitSensor Ssync1\RelDist:=0.5\MaxTime:=0.1\TimeFlag:=flag1;
```

`WaitSensor` se bude okamžitě vracet, jestliže objekt přešel 0,5 metru, ale jinak bude čekat 0,1 sek. na objekt. Jestliže žádný objekt nepřejde 0,5 metru během této 0,1 sek., instrukce bude vracet s `flag1 = TRUE`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Připojení k prvnímu objektu ve spouštěcím okně vyžaduje 50 ms. Jakmile je připojen, druhý `WaitSensor` s volitelným argumentem `\RelDist` bude trvat pouze dobu vykonávání normální instrukce RAPID.

Řešení chyb

Jestliže se během vykonávání `WaitSensor` objeví následující chyby, systémová proměnná `ERRNO` bude nastavena. Tyto chyby mohou být potom ošetřeny v chybovém handleru.

`ERR_CNV_NOT_ACT`	Senzor není aktivován.
`ERR_CNV_CONNECT`	Instrukce `WaitSensor` je již připojena.
`ERR_CNV_DROPPED`	Objekt, na který čekala instrukce `WaitSensor`, byl vyhozen jinou úlohou. (DSQC 354 Revision 2: objekt přešel spouštěcí okno)
`ERR_WAIT_MAXTIME`	Objekt nepřišel včas a není zde žádný `TimeFlag`.

Syntaxe

```
WaitSensor
  [ MechUnit ':=' ] < variable (VAR) of mecunit >
  [ '\' RelDist ':=' < expression (IN) of num > ]
  [ '\' PredTime ':=' < expression (IN) of num > ]
  [ '\' MaxTime ':=' < expression (IN) of num > ]
  [ '\' TimeFlag ':=' < variable (VAR) of bool > ] ';'
```

Související informace

Pro informace o	Viz
Shodit objekt na senzor	*DropSensor - Shodit objekt nebo senzor na str 157*
Sync k senzoru	*SyncToSensor - Synch k senzoru na str 770*
Sync k senzoru	*SyncToSensor - Synch k senzoru na str 770*
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.321 WaitSyncTask - Čekat v bodu synchronizace na jiné programové úlohy

Použití

`WaitSyncTask` se používá pro synchronizaci několika programových úloh ve zvláštním bodu každého programu. Každá programová úloha čeká, až všechny programové úlohy dosáhnou jmenovaného synchronizačního bodu.

POZNÁMKA

Instrukce `WaitSyncTask` pouze synchronizuje vykonávání programu. Pro synchronizaci jak vykonávání programu, tak i pohybů robotu musí být instrukce `Move` před `WaitSyncTask` stop bod ve všech zapojených programových úlohách. Je také možné synchronizovat vykonávání programu a pohyby robotu pomocí `WaitsyncTask \Inpos ...` ve všech zapojených programových úlohách.

VAROVÁNÍ

Aby bylo možné dosáhnout bezpečné funkčnosti synchronizace, potkávací bod (parametr `SyncID`) musí mít jedinečné jméno v každé programové úloze. Jméno musí být také stejné pro programové úlohy, které by se měly potkat v potkávacím bodě.

Základní příklady

Následující příklady názorně ukazují instrukci `WaitSyncTask`:

Viz také *Další příklady na str 958*.

Příklad 1

Příklad programu v úloze `T_ROB1`

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;

...
WaitSyncTask sync1, task_list;
...
```

Příklad 2

Příklad programu v úloze `T_ROB2`

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;

...
WaitSyncTask sync1, task_list;
...
```

Programová úloha, která jako první dosáhne `WaitSyncTask` s identitou `sync1`, čeká, až jiná programová úloha dosáhne svého `WaitSyncTask` se stejnou identitou `sync1`. Potom obě programové úlohy `T_ROB1` a `T_ROB2` pokračují ve svém vykonávání.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Argumenty

```
WaitSyncTask [\InPos] SyncID TaskList [\TimeOut]
```

`[\InPos]`

In Position

Datový typ: `switch`

Jestliže je použit tento argument, robot a externí osy musí stát v klidu, než tato programová úloha začne čekat, až jiné programové úlohy dosáhnou svého potkávacího bodu určeného v instrukci `WaitSyncTask`

`SyncID`

Synchronization identity

Datový typ: `syncident`

Proměnná, která určuje jméno bodu synchronizace (potkávacího bodu). Datový typ `syncident` je nehodnotový typ, používá se pouze jako identifikátor pro pojmenování bodu synchronizace.

Proměnná musí být definována a musí mít stejné jméno ve všech spolupracujících programových úlohách. Doporučuje se vždy definovat globální proměnnou v každé programové úloze (`VAR syncident ...`).

`TaskList`

Datový typ: `tasks`

Perzistentní proměnná, která v seznamu úloh (pole) určuje jméno (`string`) programových úloh, které by se měly potkat v bodě synchronizace se svým jménem podle argumentu `SyncID`.

Perzistentní proměnná musí být definována a musí mít stejné jméno a stejný obsah ve všech spolupracujících programových úlohách. Doporučuje se vždy definovat globální proměnnou v systému (`PERS tasks ...`).

`[\TimeOut]`

Datový typ: `num`

Max doba čekání, až jiné programové úlohy dosáhnou bodu synchronizace. Čas vypršení v sekundách (rozlišení 0,001 s). Jestliže tento argument není určen, potom bude programová úloha čekat stále.

Jestliže tento čas uběhne předtím, než všechny programové úlohy dosáhnou bodu synchronizace, bude volán chybový handler s chybovým kódem `ERR_WAITSYNCTASK`. Jestliže neexistuje chybový handler, vykonávání bude zastaveno.

Vykonávání programu

Aktuální programová úloha bude čekat na `WaitSyncTask`, až ostatní programové úlohy v `TaskList` dosáhnou stejného bodu `SyncID`. V té době bude příslušná programová úloha pokračovat s vykonáváním své další instrukce.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.321 WaitSyncTask - Čekat v bodu synchronizace na jiné programové úlohy

Multitasking
Pokračování

`WaitSyncTask` může být naprogramován mezi pohybovými instrukcemi s rohovou zónou mezi nimi. Podle načasování rovnováhy mezi programovými úlohami v době vykonávání systém může:

- při nejlepším načasování udržet všechny rohové zóny.
- při nejhorším načasování pouze udržet rohovou zónu pro programovou úlohu, která dosáhne `WaitSyncTask` jako poslední. U ostatních programových úloh budou výsledkem stop body.

Je možné vyloučit programové úlohy pro testovací účely z FlexPendantu - Panel volby úloh.

Mohou být použity následující zásady:

- Zásada 1) Vyloučit cyklicky-permanentní programovou úlohu z panelu volby úloh před spuštěním z main (po nastavení `PP` na main) - Toto odpojení bude platné během celého programového cyklu.
- Zásada 2) Vyloučit programovou úlohu dočasně z panelu výběru úloh mezi některými instrukcemi `WaitSyncTask` v programovém cyklu - Systém bude pouze provádět jiné připojené úlohy, ale bude, s chybovou zprávou, nutit uživatele připojit vyloučené programové úlohy před přejetím spolupracujícího `WaitSyncTask`.
- Zásada 3) Při běhu podle zásady 2 je možné vyloučit některý permanentní cyklus programové úlohy z panelu volby úloh pro další provádění podle zásady 1 vykonáním servisní rutiny `SkipTaskExec`.

Všimněte si, že panel volby úloh je uzamčen při běhu systému v synchronizovaných pohybech.

Další příklady

Více příkladů instrukce `WaitSyncTask` je názorně uvedeno dole.

Příklad 1

Příklad programu v úloze `T_ROB1`

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;

...
WaitSyncTask \InPos, sync1, task_list \TimeOut := 60;
...
ERROR
  IF ERRNO = ERR_WAITSYNCTASK THEN
    RETRY;
  ENDIF
```

Programová úloha `T_ROB1` čeká v instrukci `WaitSyncTask`, až její mechanické jednotky budou v pozici a potom čeká, až programová úloha `T_ROB2` dosáhne svého bodu synchronizace se stejnou identitou. Po čekání `60` sekund je zavolán chybový handler s `ERRNO` rovnou `ERR_WAITSYNCTASK`. Potom je instrukce `WaitSyncTask` volána znovu na dalších 60 s.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.321 WaitSyncTask - Čekat v bodu synchronizace na jiné programové úlohy

Multitasking
Pokračování

Řešení chyb

Jestliže se objeví vypršení časového limitu, protože `WaitSyncTask` není připraven včas, potom je systémová proměnná `ERRNO` nastavena na `ERR_WAITSYNCTASK`.

Tato chyba může být zpracována v chybovém handleru `ERROR`.

Omezení

Jestliže této instrukci předchází pohybová instrukce, tato pohybová instrukce musí být naprogramována se stop bodem (zonedata `fine`), nikoliv s průjezdným bodem. Jinak nebude možný restart po selhání napájení.

`WaitSyncTask \InPos` se nemůže provádět v RAPID rutině připojené k jakékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop,, Restart nebo Step.

Syntaxe

```
WaitSyncTask
  ['\' InPos ',']
  [ SyncID ':=' ] < variable (VAR) of syncident> ','
  [ TaskList ':=' ] < persistent array {*} (PERS) of tasks>
  [ '\' TimeOut ':=' < expression (IN) of num > ] ';'
```

Související informace

Pro informace o	Viz
Určit spolupracující programové úlohy	*tasks - Programové úlohy RAPID na str 1607*
Identita pro bod synchronizace	*syncident - Identita pro bod synchronizace na str 1603*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.322 WaitTestAndSet - Čekat, až se proměnná změní na FALSE, potom nastavit

Použití

Instrukce `WaitTestAndSet` čeká, až hodnota perzistentní proměnné `bool` bude `FALSE`. Když je hodnota proměnné `FALSE`, instrukce nastaví hodnotu na `TRUE` a pokračuje ve vykonávání. Perzistentní proměnná může být použita jako binární semafor pro synchronizace a vzájemnou vylučitelnost.

Tato instrukce má stejnou podkladovou funkčnost jako funkce `TestAndSet`, ale `WaitTestAndSet` čeká, dokud `bool` je `FALSE`, zatímco instrukce `TestAndSet` končí okamžitě.

Nedoporučuje se používat instrukci `WaitTestAndSet` v TRAP rutině. UNDO handleru nebo událostních rutinách.

Příklady zdrojů, které mohou potřebovat ochranu před přístupem ve stejnou dobu:

- Použití některých rutin RAPID s funkčními problémy při vykonávání souběžně.
- Použití FlexPendantu - Protokol operátora.

Základní příklady

Následující příklad názorně ukazuje instrukci `WaitTestAndSet`:

Viz také *Další příklady na str 961*.

Příklad 1

Programová úloha `MAIN`:

```
PERS bool tproutine_inuse := FALSE;
...
WaitTestAndSet tproutine_inuse;
TPWrite "First line from MAIN";
TPWrite "Second line from MAIN";
TPWrite "Third line from MAIN";
tproutine_inuse := FALSE;
```

Programová úloha `BACK1`:

```
PERS bool tproutine_inuse := FALSE;
...
WaitTestAndSet tproutine_inuse;
TPWrite "First line from BACK1";
TPWrite "Second line from BACK1";
TPWrite "Third line from BACK1";
tproutine_inuse := FALSE;
```

Kvůli zabránění pomíchání řádek v protokolu operátora (jedna z `MAIN` a jedna z `BACK1`) zaručuje použití funkce `WaitTestAndSet`, že všechny tři řádky z každé úlohy nejsou odděleny.

Jestliže programová úloha `MAIN` převezme semafor `WaitTestAndSet(tproutine_inuse)` jako první, programová úloha `BACK1` musí čekat, až programová úloha `MAIN` semafor opustí.

Argumenty

```
WaitTestAndSet Object
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Object`

Datový typ: `bool`

Uživatelsky definovaný datový objekt, který bude použit jako semafor. Datovým objektem musí být perzistentní proměnná `PERS`. Jestliže jsou použity `WaitTestAndSet` mezi odlišnými programovými úlohami, objektem musí být globální `PERS`.

Vykonávání programu

Tato instrukce bude v jednom nedělitelném kroku kontrolovat a nastavovat uživatelsky definovanou perzistentní proměnnou jako kódový příklad dole:

- jestliže má hodnotu `FALSE`, nastavte ji na `TRUE`
- jestliže má hodnotu `TRUE`, čekejte, až se změní na `FALSE` a potom ji nastavte na `TRUE`

```
IF Object = FALSE THEN
  Object := TRUE;
ELSE
  ! Wait until it become FALSE
  WaitUntil Object = FALSE;
  Object := TRUE;
ENDIF
```

Potom je instrukce připravena. Abyste se vyhnuli problémům, protože perzistentní proměnné si ponechávají své hodnoty, jestliže ukazatel programu PP je posunut na main, vždy nastavujte objekt semaforu na `FALSE` v událostní rutině `START`.

Další příklady

Více příkladů instrukce `WaitTestAndSet` je názorně uvedeno dole.

Příklad 1

```
PERS bool semPers:= FALSE;
...
PROC doit(...)
  WaitTestAndSet semPers;
  ...
  semPers := FALSE;
ENDPROC
```

POZNÁMKA

Jestliže vykonávání programu je zastaveno v rutině `doit` a ukazatel programu je posunut na `main`, potom proměnná `semPers` nebude resetována. Abyste se tomu vyhnuli, resetujte proměnnou `semPers` na `FALSE` v událostní rutině `START`.

Syntaxe

```
WaitTestAndSet
  [ Object ':=' ] < persistent (PERS) of bool> ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.322 WaitTestAndSet - Čekat, až se proměnná změní na FALSE, potom nastavit

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Testovat proměnnou a nastavit, jestliže není nastavena (dotaz na typ s WaitTime)	*TestAndSet - Otestovat proměnnou a nastavit, pokud není nastavena na str 1361*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.323 WaitTime - Čeká danou dobu

Použití

`WaitTime` se používá pro čekání po dané množství času. Tuto instrukci je možné také používat, dokud robot a externí osy nepřejdou do klidového stavu.

Základní příklady

Následující příklad názorně ukazuje instrukci `WaitTime`:

Viz také *Další příklady na str 963* dole.

Příklad 1

```
WaitTime 0.5;
```

Vykonávání programu čeká 0,5 sekundy.

Argumenty

```
WaitTime [\InPos] Time
```

`[\InPos]`

In Position

Datový typ: `switch`

Jestliže je použit tento argument, robot a externí osy musí být v klidovém stavu a teprve potom se začne odpočítávat doba čekání. Tento argument se může používat pouze když úloha kontroluje mechanické jednotky.

`Time`

Datový typ: `num`

Čas, vyjádřený v sekundách, kdy vykonávání programu čeká. Min hodnota 0 sek. Max hodnota nemá žádný limit. Rozlišení 0,001 sek.

Vykonávání programu

Vykonávání programu se dočasně zastaví na daný časový úsek. Zpracovávání přerušení a jiné podobné funkce jsou ale stále aktivní.

V ručním režimu, po čekání delším než 3 s se objeví okénko s upozorněním a dotazem, jestli chcete simulovat instrukci. Jestliže nechcete, aby se okénko s upozorněním objevovalo, můžete nastavit systémový parametr *SimulateMenu* na NO, viz *Technická referenční příručka - Systémové parametry*, téma *Controller*, typ *General RAPID*.

Další příklady

Více příkladů jak používat instrukci `WaitTime` je názorně uvedeno dole.

Příklad 1

```
WaitTime \InPos,0;
```

Vykonávání programu čeká, až robot a externí osy přejdou do klidového stavu.

Omezení

Argument `\Inpos` nemůže být použit společně s *SoftServo*.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.323 WaitTime - Čeká danou dobu

RobotWare - OS

Pokračování

Jestliže této instrukci předchází instrukce `Move`, potom tato instrukce `Move` musí být naprogramována se stop bodem (zonedata fine), nikoliv s průjezdným bodem. Jinak nebude možný restart po selhání napájení.

`WaitTime \Inpos` se nemůže provádět v RAPID rutině připojené k jakékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart nebo Step.

Syntaxe

```
WaitTime
  ['\' InPos ',']
  [ Time ':='] <expression (IN) of num> ';'
```

Související informace

Pro informace o	Viz
Čekání na splnění podmínky	*WaitUntil - Čeká na splnění podmínky na str 965*
Čekání na nastavení nebo resetování I/O	*WaitDI - Čeká na nastavení digitálního vstupního signálu na str 925*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.324 WaitUntil - Čeká na splnění podmínky

Použití

`WaitUntil` se používá pro čekání na splnění logické podmínky; například, může čekat na nastavení jednoho nebo několika vstupů.

Základní příklady

Následující příklad názorně ukazuje instrukci `WaitUntil`:

Viz také *Další příklady na str 967*.

Příklad 1

```
WaitUntil di4 = 1;
```

Vykonávání programu pokračuje pouze po nastavení vstupu `di4`.

Argumenty

```
WaitUntil [\InPos] Cond [\MaxTime] [\TimeFlag] [\PollRate]
     [\Visualize] [\Header] [\Message] | [\MsgArray] [\Wrap]
     [\Icon] [\Image] [\VisualizeTime]
```

`[\InPos]`

In Position

Datový typ: `switch`

Jestliže je použit tento argument, robot a externí osy musí dosáhnout stop bodu (`ToPoint` aktuální pohybové instrukce), potom může vykonávání pokračovat. Tento argument se může používat pouze když úloha kontroluje mechanické jednotky.

`Cond`

Datový typ:`bool`

Logický výraz, na který se bude čekat.

`[\MaxTime]`

Datový typ: `num`

Max povolená délka času čekání vyjádřená v sekundách. Jestliže tento čas vyprší před nastavením podmínky, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_WAIT_MAXTIME`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

`[\TimeFlag]`

Timeout Flag

Datový typ: `bool`

Výstupní parametr, který obsahuje hodnotu TRUE, jestliže max povolený čas čekání vyprší před splněním podmínky. Jestliže tento parametr je zahrnut v instrukci, potom není považováno za chybu, jestliže max čas vyprší. Tento argument je ignorován, jestliže argument `MaxTime` není zahrnut v instrukci.

`[\PollRate]`

Polling Rate

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.324 WaitUntil - Čeká na splnění podmínky
RobotWare - OS
Pokračování

Četnost výzev v sekundách pro kontrolu, jestli podmínka v argumentu `Cond` je `TRUE`. To znamená, že `WaitUntil` nejprve kontroluje podmínku hned, a jestliže není `TRUE`, potom po určené době, dokud není `TRUE`. Hodnota min četnosti výzev je 0,0004 s. Jestliže se tento argument nepoužívá, potom je výchozí četnost výzev nastavena na 0,1 s.

`[\Visualize]`

Datový typ: `switch`

Jestliže je zvoleno, aktivuje se vizualizace. Vizualizace se skládá z boxu zpráv s logickou podmínkou, která nebyla splněna, ikony, hlavičky, řádek zprávy a obrázek je zobrazen podle naprogramovaných argumentů.

`[\Header]`

Datový typ: `string`

Text hlavičky je napsán v horní části boxu zpráv. Maximum je 40 znaků. Jestliže není použit žádný argument `\Header`, bude zobrazena výchozí zpráva.

`[\Message]`

Datový typ: `string`

Na displej bude napsána jedna textová řádka. Maximum je 50 znaků.

`[\MsgArray]`

(Message Array)

Datový typ: `string`

Několik textových řádek od pole, které budou napsány na displej. Může být použit pouze jeden z parametrů `\Message` nebo `\MsgArray` ve stejném čase.

Max prostor je 5 řádek s 50 znaky na každé z nich.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 1512*.

Výchozí nastavení, žádná ikona.

`[\Image]`

Datový typ: `string`

Jméno obrázku, který by měl být použit. Chcete-li poslat své vlastní obrázky, musí být umístěny do adresáře *HOME:* v aktivním systému nebo přímo do aktivního systému.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Doporučuje se umístit soubory do adresáře *HOME:*, aby byly uloženy při provádění zálohování a obnovy.

Vyžaduje se Restart a potom FlexPendant načte obrázky.

Požadavkem na systém je, aby byl použit doplněk RobotWare *FlexPendant Interface*.

Obrázek, který má být zobrazen, může mít šířku 185 a výšku 300. Jestliže je obrázek větší, pouze 185 * 300 pixelů obrázku bude vidět od horní levé části obrázku.

Není možné určovat přesnou hodnotu velikosti obrázků nebo počet obrázků, které mohou být načteny do FlexPendantu. Závisí to na velikosti ostatních souborů načtených do FlexPendant. Vykonávání programu bude pouze pokračovat, jestliže je použit obrázek, který nebyl načten do FlexPendant.

`[\VisualizeTime]`

Datový typ: `num`

Čas čekání před tím, než by se měl objevit box zpráv na FlexPendant. Jestliže používáme argumenty `\VisualizeTime` a `\MaxTime`, čas použitý v argumentu `\MaxTime` musí být delší než čas použitý v argumentu `\VisualizeTime`.

Výchozí čas pro vizualizaci, jestliže nepoužíváme argument `\VisualizeTime` je 5 s. Minimální hodnota 1 s. Maximální hodnota nemá žádný limit. Rozlišení 0.001 s.

Vykonávání programu

Jestliže naprogramovaná podmínka není splněna při vykonávání instrukce `WaitUntil`, potom je podmínka kontrolována znovu každých 100 ms (nebo podle hodnoty určené v argumentu `Pollrate`).

Když robot čeká, čas je dohlížen a jestliže je překročena hodnota max času, program bude pokračovat, jestliže je určen `TimeFlag` nebo vyvolá chybu, jestliže tomu tak není. Jestliže je určen `TimeFlag`, bude nastaven na TRUE, jestliže čas byl překročen. Jinak bude nastaven na FALSE.

V ručním režimu, po čekání delším než 3 s se objeví okénko s upozorněním a dotazem, jestli chcete simulovat instrukci. Jestliže nechcete, aby se okénko s upozorněním objevovalo, můžete nastavit systémový parametr *SimulateMenu* na NO, viz *Technická referenční příručka - Systémové parametry*, téma *Controller*, typ *General RAPID*.

Jestliže je použit přepínač `\Visualize`, box zpráv se zobrazí na FlexPendant podle naprogramovaných argumentů. Jestliže není použit žádný argument `\Header`, zobrazí se výchozí text hlavičky. Když je vykonávání instrukce `WaitUntil` připraveno, box zpráv bude odstraněn z FlexPendant.

Nový box zpráv na úrovni TRAP přebírá prioritu od boxu zpráv na základní úrovni.

Další příklady

Více příkladů jak používat instrukci `WaitUntil` je názorně uvedeno dole.

Příklad 1

```
VAR bool timeout;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.324 WaitUntil - Čeká na splnění podmínky

RobotWare - OS

Pokračování

```
WaitUntil start_input = 1 AND grip_status = 1\MaxTime := 60
     \TimeFlag := timeout;
IF timeout THEN
  TPWrite "No start order received within expected time";
ELSE
  start_next_cycle;
ENDIF
```

Jestliže nejsou splněny dvě vstupní podmínky během `60` sekund, na displej FlexPendantu bude napsána chybová zpráva.

Příklad 2

```
WaitUntil \Inpos, di4 = 1;
```

Vykonávání programu čeká, až robot přejde do klidového stavu a až bude nastaven vstup `di4`.

Příklad 3

```
WaitUntil di4 = 1 \MaxTime:=5.5;
..
ERROR
  IF ERRNO = ERR_NORUNUNIT THEN
    TPWrite "The I/O unit is not running";
    TRYNEXT;
  ELSEIF ERRNO = ERR_WAIT_MAX THEN
    RAISE;
  ELSE
    Stop;
  ENDIF
```

Vykonávání programu čeká, až bude nastaven vstup `di4`. Jestliže jednotka I/O byla vypnuta nebo doba čekání uběhla, vykonávání pokračuje v chybovém handleru.

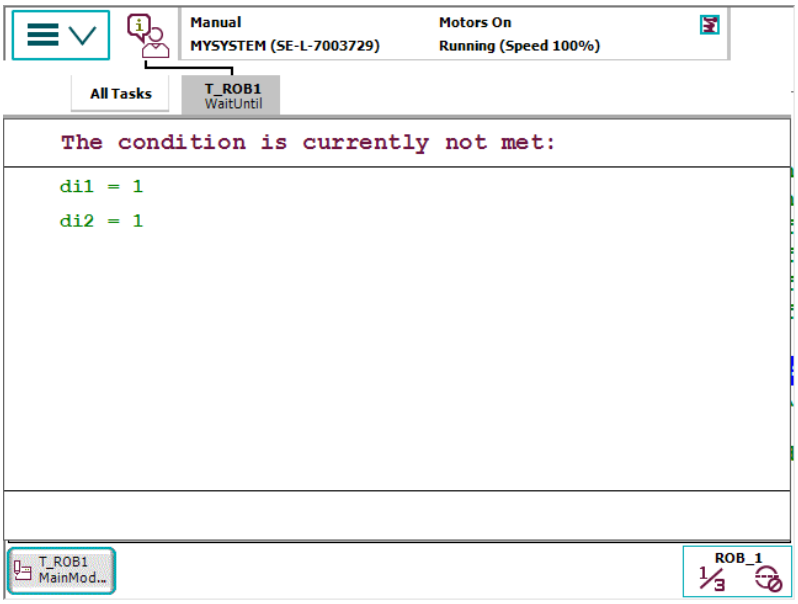
Příklad 4

```
WaitUntil di1 = 1 AND di2 = 1 \MaxTime := 60 \Visualize;
..
  ERROR
    IF ERRNO = ERR_WAIT_MAX THEN
      RAISE;
    ELSE
      Stop;
    ENDIF
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže dvě vstupní podmínky nejsou splněny během `5` sekund, na displej FlexPendantu bude napsána zpráva. Jestliže podmínky nejsou splněny během `60` sekund, vykonávání pokračuje v chybovém handleru.

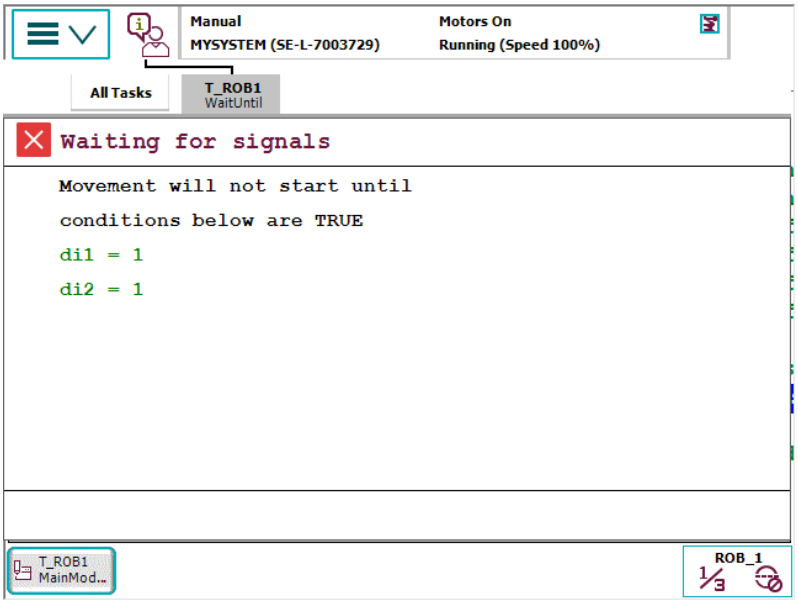


xx1600000146

Příklad 5

```
WaitUntil di1 = 1 AND di2 = 1 \Visualize \Header:="Waiting for
     signals" \MsgArray:=["Movement will not start until",
     "conditions below are TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Jestliže dvě vstupní podmínky nejsou splněny, potom bude hlavička a zpráva určená ve volitelných argumentech `\Header` a `\MsgArray` zapsána na displej FlexPendantu společně s podmínkami, které nejsou splněny.



xx1600000147

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.324 WaitUntil - Čeká na splnění podmínky

RobotWare - OS

Pokračování

Řešení chyb

Proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`. Při používání tohoto signálu je systémová proměnná `ERRNO` nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.

Jestliže existuje signál použitý v podmínce, ale neexistuje žádný kontakt s I/O jednotkou, systémová proměnná `ERRNO` je nastavena na `ERR_NORUNUNIT` a vykonávání pokračuje v chybovém handleru.

Tyto situace mohou být potom ošetřeny chybovým handlerem.

Jestliže se objevilo vypršení času (parametr `\MaxTime`) před změnou podmínky na správnou hodnotu, systémová proměnná `ERRNO` je nastavena na `ERR_WAIT_MAXTIME` a vykonávání pokračuje v chybovém handleru.

Tyto situace mohou být potom ošetřeny chybovým handlerem.

Omezení

Argument `\Inpos` nemůže být použit společně s *SoftServo*.

Jestliže této instrukci předchází instrukce `Move`, potom tato instrukce `Move` musí být naprogramována se stop bodem (zonedata fine), nikoliv s průjezdným bodem. Jinak nebude možný restart po selhání napájení.

`WaitUntil \Inpos` se nemůže provádět v RAPID rutině připojené k jakékoliv z následujících speciálních systémových událostí: PowerOn, Stop, QStop, Restart nebo Step.

`WaitUntil \Inpos` se nemůže používat společně se `StopMove` pro zjišťování, jestli byl pohyb zastaven. Instrukce `WaitUntil` může být v tomto případě odložena na neurčito. Nezjišťuje, že pohyb byl zastaven, zjišťuje, že robot a externí osy dosáhly naposledy naprogramovaného `ToPoint` (`MoveX`, `SearchX`, `TriggX`).

Syntaxe

```
WaitUntil
  ['\' InPos ',']
  [Cond ':='] <expression (IN) of bool>
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' TimeFlag ':=' <variable (VAR) of bool>]
  ['\' PollRate ':=' <expression (IN) of num>]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>] ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Čekání na nastavení nebo resetování vstupu	*WaitDI - Čeká na nastavení digitálního vstupního signálu na str 925*
Čekání po daný časový úsek	*WaitTime - Čeká danou dobu na str 963*
Výrazy	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.325 WaitWObj - Čekejte na pracovní objekt nebo dopravník

Použití

`WaitWObj` (*Wait Work Object*) připojuje k pracovnímu objektu ve spouštěcím okně na mechanické jednotce dopravníku.

Základní příklady

Následující příklad názorně ukazuje instrukci `WaitWObj`:

Viz také *Další příklady na str 973*.

Příklad 1

```
WaitWObj wobj_on_cnv1;
```

Program se připojuje k prvnímu objektu ve frontě objektů, tj. v rámci spouštěcího okna na dopravníku. Jestliže ve spouštěcím okně není žádný objekt, vykonávání čeká na objekt.

Argumenty

```
WaitWObj WObj [ \RelDist ][\MaxTime][\TimeFlag]
```

`WObj`

Work Object

Datový typ: `wobjdata`

Pohyblivý pracovní objekt (souřadnicový systém), ke kterému se vztahuje pozice robotu v instrukci. Dopravník mechanické jednotky bude určen od `ufmec` v pracovním objektu.

`[ \RelDist ]`

Relative Distance

Datový typ: `num`

Čeká na objekt, který vstoupí do spouštěcího okna a přechází za vzdálenost určenou argumentem. Jestliže pracovní objekt je již připojen, vykonávání čeká, dokud objekt nepřejde danou vzdálenost. Jestliže objekt už překročil `\RelDist`, vykonávání pokračuje.

`[\MaxTime]`

Maximum Time

Datový typ: `num`

Max povolená délka doby čekání vyjádřená v sekundách. Jestliže tento čas vyprší před připojením objektu nebo dosažením `\Reldist`, bude volán chybový handler (pokud existuje) s chybovým kódem `ERR_WAIT_MAXTIME`. Jestliže zde není žádný chybový handler, vykonávání se zastaví.

`[\TimeFlag]`

Timeout Flag

Datový typ: `bool`

Výstupní parametr, který obsahuje hodnotu `TRUE`, jestliže max povolený čas čekání vyprší před připojením objektu nebo dosažením `\Reldist`. Jestliže tento parametr

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

je zahrnut v instrukci, potom není považováno za chybu, jestliže max čas vyprší. Tento argument je ignorován, jestliže argument `MaxTime` není zahrnut v instrukci.

Vykonávání programu

Jestliže ve spouštěcím okně není žádný objekt, vykonávání programu se zastaví. Jestliže je objekt přítomný, potom je pracovní objekt připojen k dopravníku a vykonávání pokračuje.

Jestliže druhá instrukce `WaitWObj` je vydána při připojení, je vrácena chyba, pokud není použit volitelný argument `\RelDist`.

Další příklady

Více příkladů instrukce `WaitWObj` je názorně uvedeno dole.

Příklad 1

```
WaitWObj wobj_on_cnv1\RelDist:=500.0;
```

Jestliže není připojeno, čekejte na objekt, až vstoupí do spouštěcího okna, a potom čekejte na objekt, až přejde bod 500 mm na dopravníku.

Jestliže jste již připojeni k objektu, potom čekejte, až objekt přejde 500 mm.

Jestliže nejste připojeni, potom čekejte na objekt ve spouštěcím okně.

Příklad 2

```
WaitWObj wobj_on_cnv1\RelDist:=0.0;
```

Jestliže jste již připojeni, potom pokračujte s vykonáváním, protože objekt již přešel přes 0,0 mm.

Příklad 3

```
WaitWObj wobj_on_cnv1;
WaitWObj wobj_on_cnv1\RelDist:=0.0;
```

První `WaitWObj` se připojuje k objektu ve spouštěcím okně. Druhý `WaitWObj` se bude okamžitě vracet, jestliže objekt je stále připojen, ale bude čekat na další objekt, jestliže předchozí objekt se posunul za max vzdálenost nebo byl vyhozen.

Příklad 4

```
WaitWObj wobj_on_cnv1\RelDist:=500.0\MaxTime:=0.1 \Timeflag:=flag1;
```

`WaitWobj` se bude okamžitě vracet, jestliže objekt přešel 500 mm, ale jinak bude čekat 0,1 sek. na objekt. Jestliže žádný objekt nepřejde 500 mm během této 0,1 sek., instrukce se vrátí s flag1 =`TRUE`.

Omezení

Připojení k prvnímu objektu ve spouštěcím okně vyžaduje 50 ms. Jakmile je připojen, druhý `WaitWObj` s volitelným argumentem `\RelDist` bude trvat pouze dobu vykonávání normální instrukce RAPID.

Řešení chyb

Jestliže se během vykonávání instrukce `WaitWobj` objeví následující chyby, systémová proměnná `ERRNO` bude nastavena. Tyto chyby mohou být potom ošetřeny v chybovém handleru.

`ERR_CNV_NOT_ACT`	Dopravník není aktivován.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.325 WaitWObj - Čekejte na pracovní objekt nebo dopravník

Conveyor Tracking

Pokračování

ERR_CNV_CONNECT	Instrukce WaitWobj je již připojena.
ERR_CNV_DROPPED	Objekt, na který čekala instrukce WaitWobj, byl vyhozen jinou úlohou. (DSQC 354Revision 2: objekt přešel spouštěcí okno)
ERR_WAIT_MAXTIME	Objekt nepřišel včas a není zde žádný Timeflag.

Syntaxe

```
WaitWObj
  [ WObj ':='] < persistent (PERS) of wobjdata> ';'
  [ '\' RelDist ':=' < expression (IN) of num > ]
  [ '\' MaxTime ':=' <expression (IN) of num>]
  [ '\' TimeFlag ':=' <variable (VAR) of bool>] ';
```

Související informace

Pro informace o	Viz
Vyhodit pracovní objekt na dopravník	*DropWObj - Shodit pracovní objekt na dopravník na str 158*
Sledování dopravníku	*Application manual - Sledování dopravníku*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.326 WarmStart - Restartujte řadič

Použití

`WarmStart` se používá k restartu řadiče.

Systémové parametry mohou být změněny z RAPIDu s instrukcí `WriteCfgData`. Musíte restartovat řadič, aby změna měla účinek na některé systémové parametry. Restart je možné provést s touto instrukcí `WarmStart`.

Základní příklady

Následující příklad názorně ukazuje instrukci `WarmStart`:

Příklad 1

```
WriteCfgData "/MOC/MOTOR_CALIB/rob1_1","cal_offset",offset1;
WarmStart;
```

Zapisuje hodnotu `num` proměnné `offset1` jako kalibrační ofset pro osu 1 na `rob1` a generuje restart řadiče.

Vykonávání programu

`Warmstart` má okamžitý účinek a ukazatel programu je nastaven na další instrukci.

Syntaxe

```
WarmStart ';'
```

Související informace

Pro informace o	Viz
Zapsat atribut systémového parametru	*WriteCfgData - Zapisuje atribut systémového parametru na str 990*
Konfigurace	*Technická referenční příručka - Systémové parametry*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.327 WHILE - Opakuje se, dokud...

Použití

`WHILE` se používá, když více instrukcí bude opakováno, dokud výraz dané podmínky nebude vyhodnocen na hodnotu TRUE.

Tip

Jestliže je možné určit počet opakování, potom může být použita instrukce `FOR`.

Základní příklady

Následující příklad názorně ukazuje instrukci `WHILE`:

Příklad 1

```
WHILE reg1 < reg2 DO
  ...
  reg1 := reg1 + 1;
ENDWHILE
```

Opakuje instrukce v bloku `WHILE` dokud `reg1 < reg2`.

Argumenty

```
WHILE Condition DO ... ENDWHILE
```

`Condition`

Datový typ: `bool`

Podmínka, která musí být vyhodnocena na hodnotu `TRUE`, aby instrukce v bloku `WHILE` mohly být vykonány.

Vykonávání programu

1. Výraz podmínky je vyhodnocen. Jestliže výraz je vyhodnocen na hodnotu `TRUE`, jsou instrukce v bloku `WHILE` vykonány.
2. Výraz podmínky je potom vyhodnocen znovu a jestliže výsledek tohoto hodnocení je `TRUE`, potom jsou instrukce v bloku `WHILE` vykonány znovu.
3. Tento proces pokračuje, dokud výsledkem hodnocení výrazu není `FALSE`.

Iterace je potom ukončena a vykonávání programu pokračuje od instrukce po bloku `WHILE`.

Jestliže výsledek hodnocení výrazu je `FALSE` na samém začátku, potom nejsou instrukce v bloku `WHILE` vůbec vykonány a ovladač programu se přenese okamžitě k instrukci, která následuje po bloku `WHILE`.

Syntaxe

```
WHILE <conditional expression> DO
  <statement list>
ENDWHILE
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Výrazy	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Výrazy*
Opakuje se, kolikrát je stanoveno	*FOR - Opakuje, kolikrát je stanoveno na str 216*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.328 WorldAccLim - Kontrolovat zrychlení ve světovém souřadném systému

Použití

`WorldAccLim` (*World Acceleration Limitation*) se používá k limitování zrychlení/zpomalení nástroje (a užitečné zátěže) ve světovém souřadném systému.

Úplného omezení se dosáhne v bodu těžiště aktuálního nástroje, aktuální užitečné zátěže (pokud je přítomna) a montážní příruby robotu.

Základní příklady

Následující příklady názorně ukazují instrukci `WorldAccLim`:

Příklad 1

```
WorldAccLim \On := 3.5;
```

Zrychlení je omezeno na `3.5` m/s^2.

Příklad 2

```
WorldAccLim \Off;
```

Zrychlení je resetováno na maximum (výchozí nastavení).

Argumenty

```
WorldAccLim [\On]|[\Off]
```

`[ \On ]`

Datový typ: `num`

Absolutní hodnota omezení zrychlení v m/s.^{2}.

`[ \Off ]`

Datový typ: `switch`

Maximální zrychlení (výchozí nastavení).

Vykonávání programu

Omezení zrychlení se vztahují k dalšímu provedenému příkazu pohybu robotu a jsou platná až do provedení nové instrukce `WorldAccLim`.

Max zrychlení (`WorldAccLim \Off`) se nastavuje automaticky

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

Doporučuje se používat jen jeden typ omezení zrychlení. Jestliže je provedena kombinace instrukcí `WorldAccLim`, `AccSet`, a `PathAccLim` , systém snižuje zrychlení/zpomalení v následujícím pořadí:

- podle `WorldAccLim`
- podle `AccSet`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

- podle `PathAccLim`

Omezení

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Minimální povolené zrychlení je 0,1 m/s^2.

Následující modely robotů nejsou podporovány a nemohou používat instrukci `WorldAccLim`:

- IRB 340, IRB 360, IRB 540, IRB 1400, IRB 1410

Řešení chyb

Jestliže argument `On` je nastaven na hodnotu, která je příliš nízká, potom je systémová proměnná `ERRNO` nastavena na `ERR_ACC_TOO_LOW`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
WorldAccLim
  [ '\' On ':=' <expression (IN) of num>] | ['\' Off ] ';'
```

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*
Data nastavení pohybu	*motsetdata - Data nastavení pohybu na str 1533*
Snížení zrychlení	*AccSet - Omezuje zrychlení na str 19*
Omezení zrychlení podél dráhy	*PathAccLim - Omezit TCP zrychlení podél dráhy na str 454*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.329 Write - Zapisuje do souboru na základě vlastnosti nebo do sériového kanálu

Použití

`Write`ese používá pro zapisování do souboru na základě vlastností nebo do sériového kanálu. Může být zapsána hodnota konkrétních dat, stejně tak jako text.

Základní příklady

Následující příklady názorně ukazují instrukci `Write`:

Viz také *Další příklady na str 981*.

Příklad 1

```
Write logfile, "Execution started";
```

Text `Execution started` je zapsán do souboru s referenčním jménem `logfile`.

Příklad 2

```
VAR num reg1:=5;
...
Write logfile, "No of produced parts="\Num:=reg1;
```

Text `No of produced parts=5` je zapsán do souboru s referenčním jménem `logfile`.

Argumenty

```
Write IODevice String [\Num] | [\Bool] | [\Pos] | [\Orient] |
     [\Dnum] [\NoNewLine]
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) aktuálního souboru nebo sériového kanálu.

`String`

Datový typ: `string`

Text, který bude zapsán.

`[\Num]`

Numeric

Datový typ: `num`

Data, jejichž numerické hodnoty budou zapsány po textovém řetězci.

`[\Bool]`

Boolean

Datový typ: `bool`

Data, jejichž logické hodnoty budou zapsány po textovém řetězci.

`[\Pos]`

Position

Datový typ: `pos`

Data, jejichž pozice bude zapsána po textovém řetězci.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

[\Orient]

Orientation

Datový typ: orient

Data, jejichž orientace bude zapsána po textovém řetězci.

[\Dnum]

Numeric

Datový typ: dnum

Data, jejichž numerické hodnoty budou zapsány po textovém řetězci.

[\NoNewLine]

Datový typ: switch

Vynechává řádkový znak, který obvykle označuje konec textu, tj. další instrukce write bude pokračovat na stejné řádce.

Vykonávání programu

Textový řetězec je zapsán do určeného souboru nebo sériového kanálu. Posuvný řádkový znak (LF) je také zapsán, ale může být vynechán, jestliže je použit argument \NoNewLine.

Když je použit jeden z argumentů \Num, \Bool,\Pos, nebo\Orient, potom je jeho hodnota nejprve převedena do textového řetězce, předtím než je přidána do prvního řetězce. Převod z hodnoty na textový řetězec probíhá následovně:

Argument	Hodnota	Textový řetězec
\Num	23	“23”
\Num	1.141367	“1.14137”
\Bool	TRUE	“TRUE”
\Pos	[1817.3,905.17,879.11]	"[1817.3,905.17,879.11]"
\Orient	[0.96593,0,0.25882,0]	“[0.96593,0,0.25882,0]"
\Dnum	4294967295	"4294967295"

Hodnota je převedena na řetězec se standardním formátem RAPID. To znamená, v principu, 6 významných číslic. Jestliže desetinná část je méně než 0,000005 nebo více než 0,999995, číslo je zaokrouhleno na celé číslo.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu iodev bude resetován.

Další příklady

Více příkladů instrukce Write je názorně uvedeno dole.

Příklad 1

```
VAR iodev printer;
VAR num reg1:=0;
VAR num stopprod_value:=0;
...
Open "com1:", printer\Write;
stopprod_value:=stopprod;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.329 Write - Zapisuje do souboru na základě vlastnosti nebo do sériového kanálu

RobotWare - OS

Pokračování

```
WHILE stopprod_value = 0 DO
  produce_part;
  reg1:=reg1+1;
  Write printer, "Produced part="\Num:=reg1\NoNewLine;
  Write printer, " "\NoNewLine;
  Write printer, CTime();
  stopprod_value:=stopprod;
ENDWHILE
Close printer;
```

Řádka obsahující počet vyrobených kusů a čas, je při každém cyklu předána k výstupu na tiskárnu. Tiskárna je připojena k sériovému kanálu `com1`:. Vytištěná zpráva může vypadat např. takto:

`Produced part=473`	09:47:15

Omezení

Argumenty `\Num`, `\Dnum`, `\Bool`,`\Pos`, a`\Orient` jsou neslučitelné a tedy nemohou být použity současně ve stejné instrukci.

Tuto instrukci je možné používat pouze u sériových kanálů nebo souborů, které byly otevřeny pro zápis.

Řešení chyb

Jestliže se během zápisu objeví chyba, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
Write
  [ IODevice ':='] <variable (VAR) of iodev> ','
  [ String ':='] <expression (IN) of string>
  [ '\' Num ':=' <expression (IN) of num> ]
  | ['\' Bool ':=' <expression (IN) of bool> ]
  | ['\' Pos ':=' <expression (IN) of pos> ]
  | ['\' Orient ':=' <expression (IN) of orient> ]
  | ['\' Dnum ':=' <expression (IN) of dnum> ]
  [ '\' NoNewLine] ';'
```

Související informace

Pro informace o	Viz
Otevření souboru nebo sériového kanálu	*Technická referenční příručka - Přehled RAPID*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.330 WriteAnyBin - Zapisuje data do binárního sériového kanálu nebo souboru

Použití

`WriteAnyBin` (*Write Any Binary*) se používá pro zápis každého typu dat do binárního sériového kanálu nebo souboru.

Základní příklady

Následující příklad názorně ukazuje instrukci `WriteAnyBin`:

Viz také *Další příklady na str 984*.

Příklad 1

```
VAR iodev channel1;
VAR orient quat1 := [1, 0, 0, 0];
...
Open "com1:", channel1 \Bin;
WriteAnyBin channel1, quat1;
```

`orient` data `quat1` jsou zapsána do kanálu odkazovaného od `channel1`.

Argumenty

```
WriteAnyBin IODevice Data
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) binárního sériového kanálu nebo souboru, pro operaci zápisu.

`Data`

Datový typ: `ANYTYPE`

Data k zapsání.

Vykonávání programu

Tolik bajtů, kolik je požadováno pro určená data, je zapsáno do určeného binárního sériového kanálu nebo souboru.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Omezení

Tuto instrukci je možné používat pouze u sériových kanálů nebo souborů, které byly otevřeny pro binární zápis.

Data, která budou zapsána touto instrukcí `WriteAnyBin`, musí být hodnotového datového typu jako `num`, `bool`, nebo `string`. Záznam, komponent záznamu, pole nebo prvek pole těchto hodnotových datových typů mohou být také použity. Celá data nebo částečná data s polohodnotovými nebo nehodnotovými datovými typy se nemohou používat.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.330 WriteAnyBin - Zapisuje data do binárního sériového kanálu nebo souboru
RobotWare - OS
Pokračování

Řešení chyb

Jestliže se během zápisu objeví chyba, potom je systémová proměnná ERRNO nastavena na ERR_FILEACC. Tato chyba může být potom ošetřena v chybovém handleru.

Další příklady

Více příkladů instrukce WriteAnyBin je názorně uvedeno dole.

Příklad 1

```
VAR iodev channel;
VAR num input;
VAR robtarget cur_robt;

Open "com1:", channel\Bin;

! Send the control character enq
WriteStrBin channel, "\05";
! Wait for the control character ack
input := ReadBin (channel \Time:= 0.1);
IF input = 6 THEN
  ! Send current robot position
  cur_robt := CRobT(\Tool:= tool1\WObj:= wobj1);
  WriteAnyBin channel, cur_robt;
ENDIF

Close channel;
```

Aktuální pozice robotu je zapsána do binárního sériového kanálu.

Omezení

Protože WriteAnyBin-ReadAnyBin je určen pouze k odesílání interních binárních dat řadiče mezi kontrolními systémy IRC5, žádný datový protokol není vydán a data nemohou být interpretována na žádném PC.

POZNÁMKA

Vývoj ovládacího softwaru může přerušit kompatibilitu, a proto nemusí být možné používat WriteAnyBin-ReadAnyBin mezi různými verzemi softwaru RobotWare.

Syntaxe

```
WriteAnyBin
  [ IODevice ':='] <variable (VAR) of iodev> ','
  [ Data ':='] <expression (IN) of ANYTYPE> ';'
```

Související informace

Pro informace o	Viz
Otevření sériových kanálů nebo souborů	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.330 WriteAnyBin - Zapisuje data do binárního sériového kanálu nebo souboru

Pro informace o	Viz
Přečíst data z binárního sériového kanálu nebo souboru	*ReadAnyBin - Přečíst data z binárního sériového kanálu nebo souboru na str 518*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.331 WriteBin - Zapisuje do binárního sériového kanálu

Použití

`WriteBin` se používá pro zápis množství bajtů do binárního sériového kanálu.

Základní příklady

Následující příklad názorně ukazuje instrukci `WriteBin`:

Viz také *Další příklady na str 987*.

Příklad 1

```
WriteBin channel2, text_buffer, 10;
```

`10` znaků ze seznamu `text_buffer` je zapsáno do kanálu odkazovaného od `channel2`.

Argumenty

```
WriteBin IODevice Buffer NChar
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) aktuálního sériového kanálu.

`Buffer`

Datový typ: `array of num`

Seznam (pole) obsahující čísla (znaky), které budou zapsány.

`NChar`

Number of Characters

Datový typ: `num`

Počet znaků, které budou zapsány z `Buffer`.

Vykonávání programu

Určený počet čísel (znaků) v seznamu je zapsán do sériového kanálu.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Omezení

Tuto instrukci je možné používat pouze u sériových kanálů, které byly otevřeny pro binární zápis.

Řešení chyb

Jestliže se během zápisu objeví chyba, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`. Tato chyba může být potom ošetřena v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Více příkladů jak používat instrukci `WriteBin` je názorně uvedeno dole.

Příklad 1

```
VAR iodev channel;
VAR num out_buffer{20};
VAR num input;
VAR num nchar;
Open "com1:", channel\Bin;

out_buffer{1} := 5;!( enq )
WriteBin channel, out_buffer, 1;
input := ReadBin (channel \Time:= 0.1);

IF input = 6 THEN !( ack )
  out_buffer{1} := 2;!( stx )
  out_buffer{2} := 72;!( 'H' )
  out_buffer{3} := 101;!( 'e' )
  out_buffer{4} := 108;!( 'l' )
  out_buffer{5} := 108;!( 'l' )
  out_buffer{6} := 111;!( 'o' )
  out_buffer{7} := 32;!( ' ' )
  out_buffer{8} := StrToByte("w"\Char);!( 'w' )
  out_buffer{9} := StrToByte("o"\Char);!( 'o' )
  out_buffer{10} := StrToByte("r"\Char);!( 'r' )
  out_buffer{11} := StrToByte("l"\Char);!( 'l' )
  out_buffer{12} := StrToByte("d"\Char);!( 'd' )
  out_buffer{13} := 3;!( etx )
  WriteBin channel, out_buffer, 13;
ENDIF
```

Po navázání spojení (`enq`, `ack`) je textový řetězec `Hello world` (s připojenými kontrolními znaky) zapsán do sériového kanálu. Funkce `StrToByte` se používá ve stejných případech k převodu řetězce na data `byte` (`num`).

Syntaxe

```
WriteBin
  [ IODevice ':='] <variable (VAR) of iodev> ','
  [ Buffer ':='] <array {*} (IN) of num> ','
  [ NChar ':='] <expression (IN) of num> ';'
```

Související informace

Pro informace o	Viz
Otevření atd. sériových kanálů	*Technická referenční příručka - Přehled RAPID*
Převést řetězec na bajtová data	*StrToByte - Převádí řetězec na bajtová data na str 1351*
Bajtová data	*bajt - Hodnoty celého čísla 0 - 255 na str 1446*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.332 WriteBlock - Zapsat blok dat do zařízení

Použití

`WriteBlock` se používá pro zápis bloku dat do zařízení připojeného k sériovému rozhraní senzoru. Data jsou získána ze souboru.

Rozhraní senzoru komunikuje se senzory přes sériové kanály pomocí transportního protokolu RTP1.

Toto je příklad konfigurace kanálu senzoru.

```
COM_PHY_CHANNEL:
  Name "COM1:"
  Connector "COM1"
  Baudrate 19200
COM_TRP:
  Name "sen1:"
  Type "RTP1"
  PhyChannel "COM1"
```

Základní příklady

Následující příklad názorně ukazuje instrukci `WriteBlock`:

Příklad 1

```
CONST string SensorPar := "flp1:senpar.cfg";
CONST num ParBlock:= 1;

! Connect to the sensor device "sen1:" (defined in sio.cfg).
SenDevice "sen1:";

! Write sensor parameters from flp1:senpar.cfg
! to sensor datablock 1.

WriteBlock "sen1:", ParBlock, SensorPar;
```

Argumenty

```
WriteBlock device BlockNo FileName [ \TaskName ]
```

`device`

Datový typ: `string`

Jméno I/O zařízení konfigurované v sio.cfg pro použitý senzor.

`BlockNo`

Datový typ: `num`

Argument `BlockNo` se používá pro výběr datového bloku v bloku senzoru, který bude zapsán.

`FileName`

Datový typ: `string`

Argument `FileName` se používá pro výběr souboru, ze kterého budou zapsána data do datového bloku v senzoru zvoleném argumentem `BlockNo`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[ \TaskName ]`

Datový typ: `string`

Argument `TaskName` umožňuje přístup k zařízením v jiných úkolech RAPID.

Řešení chyb

Chybová konstanta (hodnota ERRNO)	Popis
SEN_NO_MEAS	Selhání měření
SEN_NOREADY	Senzor není schopen zpracovat příkaz
SEN_GENERRO	Obecná chyba senzoru
SEN_BUSY	Sběrnice senzoru
SEN_UNKNOWN	Neznámý senzor
SEN_EXALARM	Externí chyba senzoru
SEN_CAALARM	Interní chyba senzoru
SEN_TEMP	Chyba teploty senzoru
SEN_VALUE	Neplatná hodnota komunikace
SEN_CAMCHECK	Selhání kontroly senzoru
SEN_TIMEOUT	Chyba komunikace

Syntaxe

```
WriteBlock
  [ device ':=’ ] < expression(IN) of string> ','
  [ BlockNo ':=' ] < expression (IN) of num > ','
  [ FileName ':=' ] < expression (IN) of string > ','
  [ '\' TaskName ':=' < expression (IN) of string > ] ';'
```

Související informace

Pro informace o	Viz
Připojit k zařízení senzoru	*SenDevice - Připojit k zařízení senzoru na str 614*
Zapsat proměnnou senzoru	*WriteVar - Zapsat proměnnou na str 998*
Přečíst datový blok senzoru	*ReadBlock - přečíst blok dat ze zařízení na str 521*
Konfigurace komunikace senzoru	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.333 WriteCfgData - Zapisuje atribut systémového parametru

Použití

`WriteCfgData` se používá pro zápis jednoho atributu systémového parametru (konfigurační data).

Vedle zápisu jmenovitých parametrů je také možné vyhledávat a aktualizovat nepojmenované parametry

Základní příklady

Následující příklady ilustrují instrukci `WriteCfgData`. Oba tyto příklady ukazují, jak zapisovat data jmenovitých parametrů.

Příklad 1

```
VAR num offset1 := 1.2;
...
WriteCfgData "/MOC/MOTOR_CALIB/rob1_1","cal_offset",offset1;
```

Zapsán do `num` proměnné`offset1`, kalibrační ofset pro osu `1` na `rob_1`.

Příklad 2

```
VAR string io_device := "my_device";
...
WriteCfgData "/EIO/EIO_SIGNAL/process_error","Device",io_device;
```

Zapsáno do `string` proměnné`io_device`, jméno I/O zařízení, kde je definován signál `process_error`.

Argumenty

```
WriteCfgData InstancePath Attribute CfgData [\ListNo]
```

`InstancePath`

Datový typ: `string`

Určuje cestu k parametru, který má být otevřen.

Pro jmenovité parametry je formát tohoto řetězce `/DOMAIN/TYPE/ParameterName`.

Pro nejmenované parametry je formát tohoto řetězce `/DOMAIN/TYPE/Attribute/AttributeValue`.

`Attribute`

Datový typ: `string`

Jméno atributu parametru, který má být zapsán.

`CfgData`

Datový typ: `anytype`

Datový objekt, ze kterého budou nová data uložena, je přečten. Podle typu atributu jsou platné typy `bool`, `num` nebo `string`.

`[\ListNo]`

Datový typ: `num`

Proměnná držící číslo instance `Attribute` + `AttributeValue`, která má být nalezena a aktualizována.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

První výskyt `Attribute` + `AttributeValue` má číslo instance 0. Jestliže se hledá více instancí, potom bude vrácená hodnota v `\ListNo` inkrementována s 1. Jinak, jestliže už neexistují žádné další instance, potom bude vrácená hodnota -1. Předdefinovanou konstantu `END_OF_LIST` je možné použít ke kontrole, jestli je možné hledat další instance.

Vykonávání programu

Hodnota atributu určeného argumentem `Attribute` je nastavena podle hodnoty datového objektu určeného argumentem `CfgData`.

Při používání formátu `/DOMAIN/TYPE/ParameterName` v `InstancePath` mohou být získány pouze jmenovité parametry, tj. parametry, kde je prvním atributem `name`, `Name` nebo `NAME`.

U nejmenovaných parametrů použijte volitelný parametr `\ListNo` pro zvolení, do které instance se bude zapisovat hodnota atributu. Aktualizace proběhne po každém úspěšném zápisu do další dostupné instance, do které se má zapisovat.

Další příklady

Více příkladů instrukce `WriteCfgdata` je uvedeno dole. Oba tyto příklady ukazují, jak zapisovat nejmenované parametry.

Příklad 1

```
VAR num read_index;
VAR num write_index;
VAR string read_str;
...
read_index:=0;
write_index:=0;
ReadCfgData "/EIO/EIO_CROSS/Act1/do_13", "Res", read_str,
     \ListNo:=read_index;
WriteCfgData "/EIO/EIO_CROSS/Act1/do_13", "Res", "my"+read_str,
     \ListNo:=write_index;
```

Čte výsledný signál pro nejmenovaný digitální actor signál `do_13` a umisťuje jméno do řetězcové proměnné `read_str`. Potom aktualizujte jméno na `di_13` s prefixem "`my`".

V tomto příkladu má doména EIO následující cfg kód:

EIO_CROSS:

-Name "Cross_di_1_do_2" -Res "di_1" -Act1 "do_2"

-Name "Cross_di_2_do_2" -Res "di_2" -Act1 "do_2"

-Name "Cross_di_13_do_13" -Res "di_13" -Act1 "do_13"

Příklad 2

```
VAR num read_index;
VAR num write_index;
VAR string read_str;
...
read_index:=0;
write_index:=0;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.333 WriteCfgData - Zapisuje atribut systémového parametru
RobotWare - OS
Pokračování

```
WHILE read_index <> END_OF_LIST DO
  ReadCfgData "/EIO/EIO_SIGNAL/Device/USERIO", "Name", read_str,
       \ListNo:=read_index;
  IF read_index <> END_OF_LIST THEN
    WriteCfgData "/EIO/EIO_SIGNAL/Device/USERIO", "Name",
         "my"+read_str, \ListNo:=write_index;
  ENDIF
ENDWHILE
```

Přečtěte jména všech signálů definovaných pro I/O zařízení `USERIO`. Změňte jména na signálech na jména ke čtení s prefixem "`my`".

V tomto příkladu má doména EIO následující cfg kód:

```
EIO_SIGNAL:
  -Name "USERDO1" -SignalType "DO" -Device "USERIO" -DeviceMap "0"
  -Name "USERDO2" -SignalType "DO" -Device "USERIO" -DeviceMap "1"
  -Name "USERDO3" -SignalType "DO" -Device "USERIO" -DeviceMap "2"
```

Řešení chyb

Jestliže není možné najít data určená s "`InstancePath` + `Attribute`" v konfigurační databázi, potom se systémová proměnná `ERRNO` nastaví na `ERR_CFG_NOTFND`.

Jestliže datový typ pro parametr `CfgData` není totožný se skutečným datovým typem pro nalezená data určená s "`InstancePath` + `Attribute`" v konfigurační databázi, potom se systémová proměnná `ERRNO` nastaví na `ERR_CFG_ILLTYPE`..

Jestliže data pro parametr `CfgData` jsou mimo limity (max./min. hodnota), potom je systémová proměnná `ERRNO` nastavena na `ERR_CFG_LIMIT`.

Při pokusu o zápis interně zapsaných chráněných dat se potom systémová proměnná `ERRNO` nastaví na `ERR_CFG_INTERNAL`..

Jestliže proměnná v argumentu `\ListNo` má při vykonávání instrukce hodnotu mimo rozsah dostupných instancí (0 ... n), potom se `ERRNO` nastaví na `ERR_CFG_OUTOFBOUNDS`.

Tyto chyby mohou být zpracovány v chybovém handleru.

Omezení

Převod z jednotek programu RAPID (mm, stupeň, sekunda atd.) na jednotky systémových parametrů (m, radián, sekunda atd.) pro `CfgData` datového typu `num` musí provést uživatel v programu RAPID.

Musíte ručně restartovat řadič nebo vykonat instrukci `WarmStart`, aby změna měla účinek.

Při používání formátu `/DOMAIN/TYPE/ParameterName` v `InstancePath` mohou být získány pouze jmenovité parametry, tj. parametry, kde je prvním atributem `name`, `Name` nebo `NAME`.

Řetězce RAPID jsou omezeny na 80 znaků. V některých případech to může být teoreticky příliš málo pro definici `InstancePath`, `Attribute` nebo `CfgData`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Předdefinovaná data

Předdefinovaná konstanta `END_OF_LIST` s hodnotou -1 se může použít k ukončení zápisu, když není možné nalézt další instance.

Syntaxe

```
WriteCfgData
  [ InstancePath ':=' ] < expression (IN) of string > ','
  [ Attribute ':=' ] < expression (IN) of string > ','
  [ CfgData ':=' ] < expression (IN) of anytype >
  [ '\' ListNo ':=' < variable (VAR) of num >] ';'
```

Související informace

Pro informace o	Viz
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Přečíst atribut systémového parametru	*ReadCfgData - Čte atribut systémového parametru na str 523*
Získat jméno robotu v aktuální úloze	*RobName - Získat jméno TCP robotu na str 1304*
Konfigurace	*Technická referenční příručka - Systémové parametry*
Restart systému	*WarmStart - Restartujte řadič na str 975*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.334 WriteRawBytes - Zapsat data rawbytes

Použití

`WriteRawBytes` se používá pro zápis dat typu `rawbytes` do zařízení otevřeného s `Open\Bin`.

Základní příklady

Následující příklad názorně ukazuje instrukci `WriteRawBytes`:

Příklad 1

```
VAR iodev io_device;
VAR rawbytes raw_data_out;
VAR rawbytes raw_data_in;
VAR num float := 0.2;
VAR string answer;

ClearRawBytes raw_data_out;
PackDNHeader "10", "20 1D 24 01 30 64", raw_data_out;
PackRawBytes float, raw_data_out, (RawBytesLen(raw_data_out)+1)
     \Float4;

Open "/FCI1:/dsqc328_1", io_device \Bin;
WriteRawBytes io_device, raw_data_out;
ReadRawBytes io_device, raw_data_in \Time:=1;
Close io_device;

UnpackRawBytes raw_data_in, 1, answer \ASCII:=10;
```

V tomto příkladu je `raw_data_out` vyčištěno a potom zabaleno s hlavičkou DeviceNet a float s hodnotou 0.2.

Zařízení "`/FCI1/:dsqc328_1`" je otevřeno a aktuálně platná data v `raw_data_out` jsou zapsána do zařízení. Potom program čeká nejdéle 1 sek. na čtení ze zařízení, což je uloženo do `raw_data_in`.

Po zavření zařízení `/FCI1/:dsqc328_1` jsou přečtená data rozbalena jako řetězec `10` znaků a uložena do odpovědi.

Argumenty

```
WriteRawBytes IODevice RawData [\NoOfBytes]
```

`IODevice`

Datový typ: `iodev`

`IODevice` je identifikátor zařízení, do kterého budou zapsána data `RawData`.

`RawData`

Datový typ: `rawbytes`

`RawData` je datový kontejner, který bude zapsán do `IODevice`.

`[\NoOfBytes]`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`\NoOfBytes` říká, kolik bajtů `RawData` by mělo být zapsáno do `IODevice`, se začátkem na indexu 1.

Jestliže `\NoOfBytes` není přítomno, potom bude aktuální délka platných bajtů v proměnné `RawData` zapsána do zařízení IODevice.

Vykonávání programu

Během vykonávání programu jsou data zapsána do zařízení označeného `IODevice`.

Při používání `WriteRawBytes` pro příkazy aplikační sběrnice, jako je DeviceNet, potom aplikační sběrnice vždy odesílá odpověď. Odpověď musí být zpracována v RAPIDu s instrukcí `ReadRawBytes`.

Aktuální délka platných bajtů v proměnné `RawData` není změněna.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Řešení chyb

Jestliže se objeví chyba během zápisu, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`.

Tato chyba může být potom ošetřena chybovým handlerem.

Syntaxe

```
WriteRawBytes
  [ IODevice ':=' ] < variable (VAR) of iodev> ','
  [ RawData ':=' ] < variable (VAR) of rawbytes>
  [ '\' NoOfBytes ':=' < expression (IN) of num>] ';'
```

Související informace

Pro informace o	Viz
Data`rawbytes`	*rawbytes - Data raw na str 1559*
Získat délku dat `rawbytes`	*RawBytesLen - Získat délku dat rawbytes na str 1278*
Vyčistit obsah dat `rawbytes`	*ClearRawBytes - Vyčistit obsahy rawbyte dat na str 118*
Kopírovat obsah dat `rawbytes`	*CopyRawBytes - Kopírovat obsah dat rawbytes na str 137*
Zabalit hlavičku DeviceNet do dat `rawbytes`	*PackDNHeader - Zabalit hlavičku DeviceNet do dat rawbytes na str 446*
Zabalit data do dat `rawbytes`	*PackRawBytes - Zabalit data do dat rawbytes na str 449*
Načíst data `rawbytes`	*ReadRawBytes - Přečíst data rawbytes na str 530*
Rozbalit data z dat `rawbytes`	*UnpackRawBytes - Rozbalit data z rawbyte dat na str 908*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.335 WriteStrBin - Zapisuje řetězec do binárního sériového kanálu

Použití

`WriteStrBin` (*Write String Binary*) se používá pro zapsání řetězce do binárního sériového kanálu nebo binárního souboru.

Základní příklady

Následující příklad názorně ukazuje instrukci `WriteStrBin`:

Viz také *Další příklady na str 996*.

Příklad 1

```
WriteStrBin channel2, "Hello World\0A";
```

The string `"Hello World\0A"` je zapsán do kanálu odkazovaného od `channel2`. Řetězec je v tomto případě zakončen novou řádkou `\0A`. Všechny znaky a šestnáctkové hodnoty zapsané s `WriteStrBin` budou nezměněny systémem.

Argumenty

```
WriteStrBin IODevice Str
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) aktuálního sériového kanálu.

`Str`

String

Datový typ: `string`

Text, který bude zapsán.

Vykonávání programu

Textový řetězec je zapsán do určeného sériového kanálu nebo souboru.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Omezení

Tuto instrukci je možné používat pouze u sériových kanálů nebo souborů, které byly otevřeny pro binární čtení a zápis.

Řešení chyb

Jestliže se během zápisu objeví chyba, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`. Tato chyba může být potom ošetřena v chybovém handleru.

Další příklady

Více příkladů jak používat instrukci `WriteStrBin` je názorně uvedeno dole.

Příklad 1

```
VAR iodev channel;
VAR num input;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
Open "com1:", channel\Bin;

! Send the control character enq
WriteStrBin channel, "\05";
! Wait for the control character ack
input := ReadBin (channel \Time:= 0.1);
IF input = 6 THEN
  ! Send a text starting with control character stx and ending with
       etx
  WriteStrBin channel, "\02Hello world\03";
ENDIF

Close channel;
```

Po navázání spojení je textový řetězec `Hello world` (s připojenými kontrolními znaky v šestnáctkovém formátu) zapsán do binárního sériového kanálu.

Syntaxe

```
WriteStrBin
  [ IODevice ':='] <variable (VAR) of iodev> ','
  [ Str ':='] <expression (IN) of string> ';'
```

Související informace

Pro informace o	Viz
Otevření atd. sériových kanálů	*Technická referenční příručka - Přehled RAPID*
Přečíst binární řetězec	*ReadStrBin - Čte řetězec z binárního sériového kanálu nebo souboru na str 1295*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.336 WriteVar - Zapsat proměnnou

Použití

`WriteVar` se používá pro zápis proměnné do zařízení připojeného k sériovému rozhraní senzoru.

Rozhraní senzoru komunikuje se senzory přes sériové kanály pomocí transportního protokolu RTP1.

Toto je příklad konfigurace kanálu senzoru.

```
COM_PHY_CHANNEL:
  Name "COM1:"
  Connector "COM1"
  Baudrate 19200
COM_TRP:
  Name "sen1:"
  Type "RTP1"
  PhyChannel "COM1"
```

Základní příklady

Následující příklad názorně ukazuje instrukci `WriteVar`:

Příklad 1

```
! Define variable numbers
CONST num SensorOn := 6;
CONST num XCoord := 8;
CONST num YCoord := 9;
CONST num ZCoord := 10;
VAR pos SensorPos;

! Connect to the sensor device“ sen1:” (defined in sio.cfg).
SenDevice "sen1:";

! Request start of sensor meassurements
WriteVar "sen1:", SensorOn, 1;

! Read a cartesian position from the sensor.
SensorPos.x := ReadVar "sen1:", XCoord;
SensorPos.y := ReadVar "sen1:", YCoord;
SensorPos.z := ReadVar "sen1:", ZCoord;

! Stop sensor
WriteVar "sen1:", SensorOn, 0;
```

Argumenty

```
WriteVar device VarNo VarData [ \TaskName ]
```

`device`

Datový typ: `string`

Jméno I/O zařízení konfigurované v sio.cfg pro použitý senzor.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`VarNo`

Datový typ: `num`

Argument `VarNo` se používá pro výběr proměnné senzoru.

`VarData`

Datový typ: `num`

Argument `VarData` definuje data, která budou zapsána do proměnné vybrané argumentem `VarNo`.

`[ \TaskName ]`

Datový typ: `string`

Argument `TaskName` umožňuje přístup k zařízením v jiných úkolech RAPID.

Řešení chyb

Hodnota chybové konstanty (ERRNO)	Popis
SEN_NO_MEAS	Selhání měření
SEN_NOREADY	Senzor není schopen zpracovat příkaz
SEN_GENERRO	Obecná chyba senzoru
SEN_BUSY	Snímač je zaneprázdněn
SEN_UNKNOWN	Neznámý senzor
SEN_EXALARM	Externí chyba senzoru
SEN_CAALARM	Interní chyba senzoru
SEN_TEMP	Chyba teploty senzoru
SEN_VALUE	Neplatná hodnota komunikace
SEN_CAMCHECK	Selhání kontroly senzoru
SEN_TIMEOUT	Chyba komunikace

Syntaxe

```
WriteVar
  [ device ':=' ] < expression (IN) of string> ','
  [ VarNo ':=' ] < expression (IN) of num > ','
  [ VarData ':=' ] < expression (IN) of num > ','
  [ '\' TaskName ':=' < expression (IN) of string > ] ';'
```

Související informace

Pro informace o	Viz
Připojit k zařízení senzoru	*SenDevice - Připojit k zařízení senzoru na str 614*
Přečíst proměnnou senzoru	*ReadVar - Přečíst proměnnou ze zařízení na str 1297*
Zapsat datový blok senzoru	*WriteBlock - Zapsat blok dat do zařízení na str 988*
Přečíst datový blok senzoru	*ReadBlock - přečíst blok dat ze zařízení na str 521*
Konfigurace komunikace senzoru	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.337 WZBoxDef - Definovat světovou zónu tvaru box

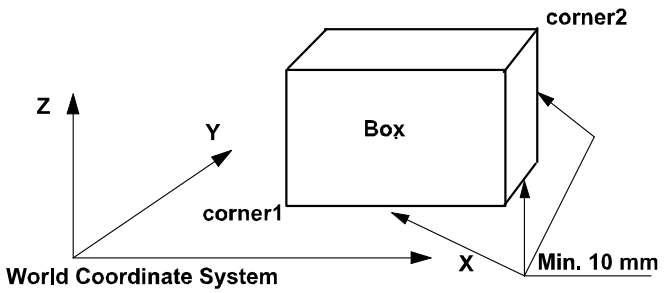
Použití

`WZBoxDef` (*World Zone Box Definition*) se používá k definování světové zóny, která má tvar rovného boxu se všemi jeho stranami souběžnými s osami světového souřadného systému.

Základní příklady

Následující příklad názorně ukazuje instrukci `WZBoxDef`:

Příklad 1



xx0500002205

```
VAR shapedata volume;
CONST pos corner1:=[200,100,100];
CONST pos corner2:=[600,400,400];
...
WZBoxDef \Inside, volume, corner1, corner2;
```

Definujte rovný box se souřadnicemi souběžnými s osami světového souřadného systému a definovanými protějšími rohy `corner1` a `corner2`.

Argumenty

```
WZBoxDef [\Inside] | [\Outside] Shape LowPoint HighPoint
```

`[\Inside]`

Datový typ: `switch`

Definovat objem uvnitř boxu.

`[\Outside]`

Datový typ: `switch`

Definovat objem vně boxu (inverzní objem).

Musí být určen jeden z argumentů `\Inside` nebo `\Outside`.

`Shape`

Datový typ: `shapedata`

Proměnná pro uložení definovaného objemu (privátní data pro systém).

`LowPoint`

Datový typ: `pos`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pozice (x,y,z) v mm definující jeden dolní roh boxu.

`HighPoint`

Datový typ: `pos`

Pozice (x,y,z) v mm definující roh úhlopříčně protější k předchozímu.

Vykonávání programu

Definice boxu je uložena v proměnné typu `shapedata` (argument `Shape`) pro pozdější použití v instrukcích `WZLimSup` nebo `WZDOSet`.

Omezení

Pozice `LowPoint` a `HighPoint` musí být platné pro protější rohy (s různými souřadnými hodnotami x, y a z).

Jestliže robot je použit pro zdůraznění `LowPoint` nebo `HighPoint`, potom musí být pracovní objekt `wobj0` aktivní (použijte komponent `trans` v `robtarget` např. `p1.trans` jako argument).

Syntaxe

```
WZBoxDef
  [ ['\' Inside] | ['\' Outside] ',']
  [ LowPoint ':='] <expression (IN) of pos> ','
  [ Shape ':='] <variable (VAR) of shapedata> ','
  [ HighPoint ':='] <expression (IN) of pos> ';'
```

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Definovat světovou zónu tvaru koule	*WZSphDef - Definovat světovou zónu tvaru koule na str 1025*
Definovat světovou zónu tvaru válce	*WZCylDef - Definovat světovou zónu tvaru válce na str 1002*
Definovat světovou zónu pro home spoje	*WZHomeJointDef - Definovat světovou zónu pro home spoje na str 1015*
Definovat světovou zónu pro limit spoje	*WZLimJointDef - Definovat světovou zónu pro omezení ve spojích na str 1018*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Aktivovat nastavení digitálního výstupu světové zóny	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.338 WZCylDef - Definovat světovou zónu tvaru válce

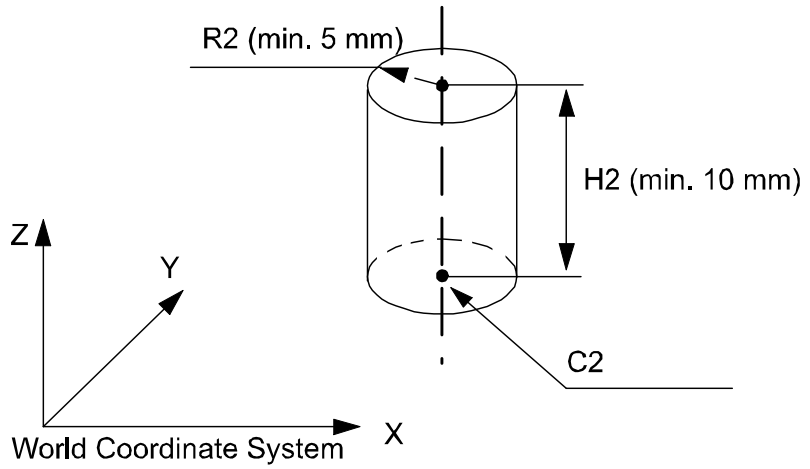
Použití

`WZCylDef` (*World Zone Cylinder Definition*) is se používá k definování světové zóny, která má tvar válce s osou válce souběžnou s osou z světového souřadného systému.

Základní příklady

Následující příklad názorně ukazuje instrukci `WZCylDef`:

Příklad 1



xx0500002206

```
VAR shapedata volume;
CONST pos C2:=[300,200,200];
CONST num R2:=100;
CONST num H2:=200;
...
WZCylDef \Inside, volume, C2, R2, H2;
```

Definovat válec se středem dolního kruhu v `C2`, poloměr`R2`, a výškou `H2`.

Argumenty

```
WZCylDef [\Inside] | [\Outside] Shape CentrePoint Radius Height
```

`[\Inside]`

Datový typ: `switch`

Definovat objem uvnitř válce.

`[\Outside]`

Datový typ: `switch`

Definovat objem vně válce (inverzní objem).

Musí být určen jeden z argumentů `\Inside` nebo `\Outside`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Shape`

Datový typ: `shapedata`

Proměnná pro uložení definovaného objemu (privátní data pro systém).

`CentrePoint`

Datový typ: `pos`

Pozice (x,y,z) v mm definující střed jednoho kruhového konce válce.

`Radius`

Datový typ: `num`

Poloměr válce v mm.

`Height`

Datový typ: `num`

Výška válce v mm. Jestliže je kladná (směr +z), argument `CentrePoint` je středem dolního konce válce (jako v příkladu nahoře). Jestliže je záporná (směr -z), potom argument `CentrePoint` je středem horního konce válce.

Vykonávání programu

Definice válce je uložena v proměnné typu `shapedata` (argument `Shape`) pro pozdější použití v instrukcích `WZLimSup` nebo `WZDOSet`.

Omezení

Jestliže robot je použit pro zdůraznění `CentrePoint`, potom musí být pracovní objekt `wobj0` aktivní (použijte komponent `trans` v `robtarget` např. `p1.trans` jako argument).

Syntaxe

```
WZCylDef
  [ '\' Inside] | [ '\' Outside] ','
  [ Shape ':='] <variable (VAR) of shapedata> ','
  [ CentrePoint ':='] <expression (IN) of pos> ','
  [ Radius ':='] <expression (IN) of num> ','
  [ Height ':='] <expression (IN) of num> ';'
```

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Definovat světovou zónu tvaru boxu	*WZBoxDef - Definovat světovou zónu tvaru box na str 1000*
Definovat světovou zónu tvaru koule	*WZSphDef - Definovat světovou zónu tvaru koule na str 1025*
Definovat světovou zónu pro home spoje	*WZHomeJointDef - Definovat světovou zónu pro home spoje na str 1015*
Definovat světovou zónu pro limit spoje	*WZLimJointDef - Definovat světovou zónu pro omezení ve spojích na str 1018*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.338 WZCylDef - Definovat světovou zónu tvaru válce

World Zones

Pokračování

Pro informace o	Viz
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Aktivovat nastavení digitálního výstupu světové zóny	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.339 WZDisable - Deaktivovat dohled dočasné světové zóny

Použití

`WZDisable` (*World Zone Disable*) se používá pro deaktivaci dohledu dočasné světové zóny dříve definované k zastavení pohybu nebo nastavení výstupu.

Základní příklady

Následující příklad názorně ukazuje instrukci `WZDisable`:

Příklad 1

```
VAR wztemporary wzone;
...
PROC...
  WZLimSup \Temp, wzone, volume;
  MoveL p_pick, v500, z40, tool1;
  WZDisable wzone;
  MoveL p_place, v200, z30, tool1;
ENDPROC
```

Při pohybu k `p_pick` je pozice TCP robotu kontrolována, aby se nedostal do určeného objemu `wzone`. Tento dohled se neprovádí při přechodu do `p_place`.

Argumenty

```
WZDisable WorldZone
```

`WorldZone`

Datový typ: `wztemporary`

Proměnná nebo perzistentní proměnná typu `wztemporary`, která obsahuje identitu světové zóny, která bude aktivována.

Vykonávání programu

Dočasná světová zóna je deaktivována. To znamená, že dohled nad TCP robotu ve vztahu k odpovídajícímu objemu je dočasně zastaven. Může být znovu aktivován přes instrukci `WZEnable`.

Omezení

Pouze dočasná světová zóna může být deaktivována. Stacionární světová zóna je vždy aktivní.

Syntaxe

```
WZDisable
  [ WorldZone ':='] <variable or persistent (INOUT) of wztemporary>
       ';'
```

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.339 WZDisable - Deaktivovat dohled dočasné světové zóny

World Zones

Pokračování

Pro informace o	Viz
Data dočasné světové zóny	*wztemporary - Data dočasné světové zóny na str 1640*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Aktivovat světovou zónou nastavený digitální výstup	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*
Aktivovat světovou zónu	*WZEnable - Aktivovat dohled dočasné světové zóny na str 1011*
Vymazat světovou zónu	*WZFree - Vymazat dohled dočasné světové zóny na str 1013*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.340 WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu

Použití

`WZDOSet` (*World Zone Digital Output Set*) se používá pro definování činnosti a aktivaci světové zóny pro dohled nad pohyby robotu.

Po vykonání této instrukce, když TCP robotu/externích os (zóna ve spojích) je uvnitř definované světové zóny nebo se k ní blíží, je nastaven digitální výstupní signál na určenou hodnotu.

Základní příklady

Následující příklad názorně ukazuje instrukci `WZDOSet`:

Viz také *Další příklady na str 1008*.

Příklad 1

```
VAR wztemporary service;

PROC zone_output()
  VAR shapedata volume;
  CONST pos p_service:=[500,500,700];
  ...
  WZSphDef \Inside, volume, p_service, 50;
  WZDOSet \Temp, service \Inside, volume, do_service, 1;
ENDPROC
```

Definice dočasné světové zóny `service` v aplikačním programu, který nastavuje signál `do_service`, když TCP robotu je uvnitř definované sféry během vykonávání programu nebo při ručním posuvu (jogging).

Argumenty

```
WZDOSet [\Temp] | [\Stat] WorldZone [\Inside] | [\Before] Shape
     Signal SetValue
```

`[\Temp]`

Temporary

Datový typ: `switch`

Světová zóna k definování je dočasná světová zóna.

`[\Stat]`

Stationary

Datový typ: `switch`

Světová zóna k definování je stacionární světová zóna.

Musí být určen jeden z argumentů `\Temp` nebo `\Stat`.

`WorldZone`

Datový typ: `wztemporary` nebo `wzstationary`

Proměnná nebo perzistentní proměnná, která bude aktualizována s identitou (numerická hodnota) světové zóny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.340 WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu
World Zones
Pokračování

Při používání přepínače `\Temp`, musí být datový typ `wztemporary`. Při používání přepínače `\Stat`, datový typ musí být `wzstationary`.

`[\Inside]`

Datový typ: `switch`

Digitální výstupní signál bude nastaven, když TCP robotu nebo určené osy jsou uvnitř definovaného objemu.

`[\Before]`

Datový typ: `switch`

Digitální výstupní signál bude nastaven předtím, než TCP robotu nebo určené osy dosáhnou definovaného objemu (co nejdříve před objemem).

Musí být určen jeden z argumentů `\Inside` nebo `\Before`.

`Shape`

Datový typ: `shapedata`

Proměnná, která definuje objem světové zóny.

`Signal`

Datový typ: `signaldo`

Jméno digitálního výstupního signálu, který bude změněn.

Jestliže se používá stacionární světová zóna, signál musí být zapsán jako chráněný pro přístup od uživatele (RAPID, FP). Nastavte Access Level pro signál v systémových parametrech nebo určených osách.

`SetValue`

Datový typ: `dionum`

Požadovaná hodnota signálu (0 nebo 1), když TCP robotu je uvnitř objemu nebo právě před vstupem do objemu.

Ve stavu mimo nebo skoro mimo objemu je signál nastaven na opačnou hodnotu.

Vykonávání programu

Definovaná světová zóna je aktivována. Od tohoto momentu je TCP robotu (nebo pozice robot/externí spoj) pod dohledem a výstup bude nastaven, když pozice TCP robotu (nebo pozice robot/externí spoj) je uvnitř objemu (`\Inside`) nebo přichází blízko ke hranici objemu (`\Before`).

Při používání `WZHomeJointDef` nebo `WZLimJointDef` společně s `WZDOSet` je digitální výstupní signál nastaven pouze když všechny aktivní osy s dohledem prostoru spoje jsou před nebo uvnitř prostoru spoje.

Další příklady

Více příkladů jak používat instrukci `WZDOSet` je názorně uvedeno dole.

Příklad 1

```
VAR wztemporary home;
VAR wztemporary service;
PERS wztemporary equip1:=[0];
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
PROC main()
  ...
  ! Definition of all temporary world zones
  zone_output;
  ...
  ! equip1 in robot work area
  WZEnable equip1;
  ...
  ! equip1 out of robot work area
  WZDisable equip1;
  ...
  ! No use for equip1 any more
  WZFree equip1;
  ...
ENDPROC

PROC zone_output()
  VAR shapedata volume;
  CONST pos p_home:=[800,0,800];
  CONST pos p_service:=[800,800,800];
  CONST pos p_equip1:=[-800,-800,0];
  ...
  WZSphDef \Inside, volume, p_home, 50;
  WZDOSet \Temp, home \Inside, volume, do_home, 1;
  WZSphDef \Inside, volume, p_service, 50;
  WZDOSet \Temp, service \Inside, volume, do_service, 1;
  WZCylDef \Inside, volume, p_equip1, 300, 1000;
  WZLimSup \Temp, equip1, volume;
  ! equip1 not in robot work area
  WZDisable equip1;
ENDPROC
```

Definice dočasných světových zón `home` a `service` v aplikačním programu, který nastavuje signály `do_home` a `do_service`, když robot je uvnitř sféry `home` nebo `service` během vykonávání programu nebo při ručním posuvu (jogging).

Také definice dočasné světové zóny `equip1`, která je aktivní pouze v části programu robotu, když `equip1` je uvnitř pracovní oblasti pro robot. V té době se robot zastaví před vstupem do objemu `equip1` jak během vykonávání programu, tak i během ručního posuvu (jogging). `equip1` může být vypnut nebo zapnut z jiných programových úloh pomocí hodnoty perzistentní proměnné `equip1`.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.340 WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu

World Zones
Pokračování

Omezení

Světová zóna nemůže být znovu definována pomocí stejné proměnné v argumentu `WorldZone`.

Stacionární světová zóna nemůže být deaktivována, znovu aktivována nebo vymazána v programu RAPID.

Dočasná světová zóna může být deaktivována (`WZDisable`), znovu aktivována (`WZEnable`) nebo vymazána (`WZFree`)) v programu RAPID.

Syntaxe

```
WZDOSet
  [ ['\' Temp] | ['\' Stat] ',']
  [ WorldZone ':='] <variable or persistent (INOUT) of wztemporary>
  [ '\' Inside] | ['\' Before] ','
  [ Shape ':='] <variable (VAR) of shapedata> ','
  [ Signal ':='] <variable (VAR) of signaldo> ','
  [ SetValue ':='] <expression (IN) of dionum> ';'
```

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Dočasná světová zóna	*wztemporary - Data dočasné světové zóny na str 1640*
Stacionární světová zóna	*wzstationary - Data stacionární světové zóny na str 1638*
Definovat světovou zónu tvaru rovného boxu	*WZBoxDef - Definovat světovou zónu tvaru box na str 1000*
Definovat světovou zónu tvaru koule	*WZSphDef - Definovat světovou zónu tvaru koule na str 1025*
Definovat světovou zónu tvaru válce	*WZCylDef - Definovat světovou zónu tvaru válce na str 1002*
Definovat světovou zónu pro home spoje	*WZHomeJointDef - Definovat světovou zónu pro home spoje na str 1015*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Úroveň přístupu signálu	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.341 WZEnable - Aktivovat dohled dočasné světové zóny

Použití

`WZEnable` (*World Zone Enable*) se používá pro novou aktivaci dohledu dočasné světové zóny, dříve definované buď k zastavení pohybu nebo nastavení výstupu.

Základní příklady

Následující příklad názorně ukazuje instrukci `WZEnable`:

Příklad 1

```
VAR wztemporary wzone;
...
PROC ...
  WZLimSup \Temp, wzone, volume;
  MoveL p_pick, v500, z40, tool1;
  WZDisable wzone;
  MoveL p_place, v200, z30, tool1;
  WZEnable wzone;
  MoveL p_home, v200, z30, tool1;
ENDPROC
```

Při pohybu k `p_pick` je pozice TCP robotu kontrolována, aby se nedostal do určeného objemu `wzone`. Tento dohled se neprovádí při přechodu do `p_place`, ale je znovu aktivován před přechodem do `p_home`.

Argumenty

```
WZEnable WorldZone
```

`WorldZone`

Datový typ: `wztemporary`

Proměnná nebo perzistentní proměnná typu `wztemporary`, která obsahuje identitu světové zóny, která bude aktivována.

Vykonávání programu

Dočasná světová zóna je znovu aktivována. Všimněte si, že světová zóna je automaticky aktivována, když je vytvořena. Musí být znovu aktivována pouze v případě, že byla předtím deaktivována z `WZDisable`.

Omezení

Pouze dočasná světová zóna může být deaktivována a znovu aktivována. Stacionární světová zóna je vždy aktivní.

Syntaxe

```
WZEnable
  [ WorldZone ':='] <variable or persistent (INOUT) of wztemporary>
       ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.341 WZEnable - Aktivovat dohled dočasné světové zóny

World Zones

Pokračování

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Data dočasné světové zóny	*wztemporary - Data dočasné světové zóny na str 1640*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Aktivovat světovou zónou nastavený digitální výstup	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*
Deaktivovat světovou zónu	*WZDisable - Deaktivovat dohled dočasné světové zóny na str 1005*
Vymazat světovou zónu	*WZFree - Vymazat dohled dočasné světové zóny na str 1013*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.342 WZFree - Vymazat dohled dočasné světové zóny

Použití

`WZFree` (*World Zone Free*) se používá pro vymazání definice dočasné světové zóny dříve definované k zastavení pohybu nebo nastavení výstupu.

Základní příklady

Následující příklad názorně ukazuje instrukci `WZFree`:

Příklad 1

```
VAR wztemporary wzone;
...
PROC ...
  WZLimSup \Temp, wzone, volume;
  MoveL p_pick, v500, z40, tool1;
  WZDisable wzone;
  MoveL p_place, v200, z30, tool1;
  WZEnable wzone;
  MoveL p_home, v200, z30, tool1;
  WZFree wzone;
ENDPROC
```

Při pohybu k `p_pick` je pozice TCP robotu kontrolována, aby se nedostal do určeného objemu `wzone`. Tento dohled se neprovádí při přechodu do `p_place`, ale je znovu aktivován před přechodem do `p_home`. Když je dosaženo této pozice, definice světové zóny je vymazána.

Argumenty

```
WZFree WorldZone
```

`WorldZone`

Datový typ: `wztemporary`

Proměnná nebo perzistentní proměnná typu `wztemporary`, která obsahuje identitu světové zóny, která bude vymazána.

Vykonávání programu

Dočasná světová zóna je nejprve deaktivována a potom je její definice vymazána.

Jakmile je vymazána, už není možné dočasnou světovou zónu znovu aktivovat nebo deaktivovat.

Omezení

Pouze dočasná světová zóna může být deaktivována, znovu aktivována nebo vymazána. Stacionární světová zóna je vždy aktivní.

Syntaxe

```
WZFree
  [ WorldZone ':='] <variable or persistent (INOUT) of wztemporary>
       ';'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.342 WZFree - Vymazat dohled dočasné světové zóny

World Zones

Pokračování

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Data dočasné světové zóny	*wztemporary - Data dočasné světové zóny na str 1640*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Aktivovat světovou zónou nastavený digitální výstup	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*
Deaktivovat světovou zónu	*WZDisable - Deaktivovat dohled dočasné světové zóny na str 1005*
Aktivovat světovou zónu	*WZEnable - Aktivovat dohled dočasné světové zóny na str 1011*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.343 WZHomeJointDef - Definovat světovou zónu pro home spoje

Použití

`WZHomeJointDef` (*World Zone Home Joint Definition*) se používá pro definování světové zóny v souřadnicích spojů pro robot a externí osy, které budou použity jako pozice HOME nebo SERVICE.

Základní příklady

Následující příklad názorně ukazuje instrukci `WZHomeJointDef`:

Příklad 1

```
VAR wzstationary home;
...
PROC power_on()
  VAR shapedata joint_space;
  CONST jointtarget home_pos := [ [ 0, 0, 0, 0, 0, -45], [ 0, 9E9,
       9E9, 9E9, 9E9, 9E9] ];
  CONST jointtarget delta_pos := [ [ 2, 2, 2, 2, 2, 2], [ 5, 9E9,
       9E9, 9E9, 9E9, 9E9] ];
  ...
  WZHomeJointDef \Inside, joint_space, home_pos, delta_pos;
  WZDOSet \Stat, home \Inside, joint_space, do_home, 1;
ENDPROC
```

Definice a aktivace stacionární světové zóny `home`, která nastavuje signál `do_home to 1`, , když všechny osy robotu a externí osy `extax.eax_a` jsou v pozici spoje `home_pos` (v +/- `delta_pos` pro každou osu) během vykonávání programu a ručního posuvu (jogging). Proměnná `joint_space` datového typu `shapedata` se používá pro přenos dat z instrukce `WZHomeJointDef` do instrukce `WZDOSet`.

Argumenty

```
WZHomeJointDef [\Inside] | [\Outside] Shape MiddleJointVal
     DeltaJointVal
```

`[\Inside]`

Datový typ: `switch`

Definovat prostor spoje uvnitř `MiddleJointVal` +/- `DeltaJointVal`.

`[\Outside]`

Datový typ: `switch`

Definovat prostor spoje mimo `MiddleJointVal` +/- `DeltaJointVal` (inverzní prostor spoje).

`Shape`

Datový typ: `shapedata`

Proměnná pro uložení definovaného prostoru spoje (privátní data pro systém).

`MiddleJointVal`

Datový typ: `jointtarget`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.343 WZHomeJointDef - Definovat světovou zónu pro home spoje
World Zones
Pokračování

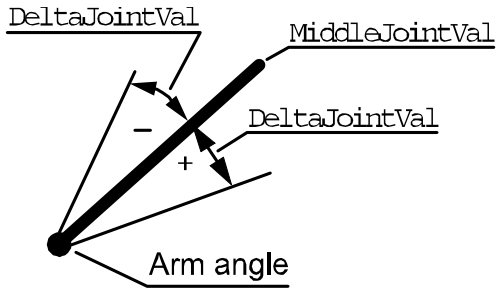
Pozice souřadnic spoje pro střed prostoru spoje k definování. Určuje pro každou osu robotu a externí osu (stupně u rotačních os a mm u lineárních os). Určuje v absolutních spojích (nikoliv v ofsetovém souřadném systému `EOffsSet-EOffsOn` pro externí osy). **Hodnota 9E9 u některých os znamená, že osa by neměla být dohlížena.** Neaktivní externí osa také dává 9E9 v době programování.

`DeltaJointVal`

Datový typ: `jointtarget`

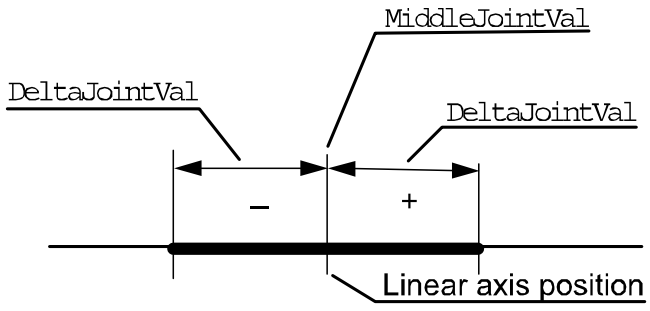
Pozice +/- delta v souřadnicích spoje od středu prostoru spoje. Hodnota musí být větší než 0 u všech os k dohlížení.

Následující obrázek ukazuje definici prostoru spoje pro rotační osu.



xx0500002208

Následující obrázek ukazuje definici prostoru spoje pro lineární osu.



xx0500002209

Vykonávání programu

Definice prostoru spoje je uložena v proměnné typu `shapedata` (argument `Shape`) pro pozdější použití v instrukcích `WZLimSup` nebo `WZDOSet`.

Při používání `WZHomeJointDef` společně s `WZDOSet` je digitální výstupní signál nastaven pouze když všechny aktivní osy s dohledem prostoru spoje jsou před nebo uvnitř prostoru spoje.

Při použití `WZHomeJointDef` s vnějším prostorem spoje (argument `\Outside`) společně s `WZLimSup` je robot zastaven ihned, jakmile jedna aktivní osa s dohledem prostoru spoje dosáhne prostoru spoje.

Při použití `WZHomeJointDef` s vnitřním prostorem spoje (argument `\Inside`), společně s `WZLimSup`) je robot zastaven, jakmile poslední aktivní osa s dohledem prostoru svaru dosáhne prostoru spoje. To znamená, že jedna nebo několik os, ale nikoliv všechny aktivní a sledované osy, mohou být uvnitř prostoru spoje ve stejnou dobu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Při vykonávání instrukce `ActUnit` nebo `DeactUnit` pro aktivaci nebo deaktivaci mechanických jednotek bude aktualizován status dohledu pro pozici HOME nebo omezení pracovní oblasti.

Omezení

xx0100000002

Pouze aktivní mechanické jednotky a jejich aktivní osy v době aktivace světové zóny (s instrukcí `WZDOSet` resp. `WZLimSup`) jsou zahrnuty do dohledu pozice HOME do omezení pracovní oblasti. Kromě toho, mechanické jednotky a jejich osy musí být ještě aktivní při pohybu programu nebo ručním posuvu (jogging), aby mohly být sledovány.

Například, jestliže jedna osa s dohledem je mimo svoji HOME pozici spoje, ale je deaktivována, potom to nechrání digitální výstupní signál pro pozici spoje HOME před nastavením, jestliže všechny ostatní aktivní osy s dohledem prostoru spoje jsou uvnitř HOME pozice prostoru spoje. Při nové aktivaci této osy bude zahrnuta do dohledu a systém robotu potom bude mimo pozici spojeHOME a digitální výstup bude resetován.

Syntaxe

```
WZHomeJointDef
  [ ['\' Inside] | ['\' Outside] ',']
  [ Shape ':='] <variable (VAR) of shapedata> ','
  [ MiddleJointVal ':='] <expression (IN) of jointtarget> ','
  [ DeltaJointVal ':='] <expression (IN) of jointtarget> ';'
```

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Definovat světovou zónu tvaru boxu	*WZBoxDef - Definovat světovou zónu tvaru box na str 1000*
Definovat světovou zónu tvaru válce	*WZCylDef - Definovat světovou zónu tvaru válce na str 1002*
Definovat světovou zónu tvaru koule	*WZSphDef - Definovat světovou zónu tvaru koule na str 1025*
Definovat světovou zónu pro limit spoje	*WZLimJointDef - Definovat světovou zónu pro omezení ve spojích na str 1018*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Aktivovat nastavení digitálního výstupu světové zóny	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.344 WZLimJointDef - Definovat světovou zónu pro omezení ve spojích

Použití

`WZLimJointDef` (*World Zone Limit Joint Definition*) se používá pro definování světové zóny v souřadnicích spojů pro robot a externí osy, které budou použity pro omezení v pracovní oblasti.

S `WZLimJointDef` je možné limitovat pracovní oblast pro každý robot a externí osy v programu RAPID, kromě omezení, které může být provedeno se systémovými parametry *Motion - Arm - robx_y - Upper Joint Bound ... Lower Joint Bound*.

Základní příklady

Následující příklad názorně ukazuje instrukci `WZLimJointDef`:

Příklad 1

```
VAR wzstationary work_limit;
  ...
  PROC power_on()
    VAR shapedata joint_space;
    CONST jointtarget low_pos:= [ [ -90, 9E9, 9E9, 9E9, 9E9, 9E9],
         [ -1000, 9E9, 9E9, 9E9, 9E9, 9E9]];
    CONST jointtarget high_pos := [ [ 90, 9E9, 9E9, 9E9,9E9, 9E9],
         [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
    ...
    WZLimJointDef \Outside, joint_space, low_pos, high_pos;
    WZLimSup \Stat, work_limit, joint_space;
  ENDPROC
```

Definice a aktivace stacionární světové zóny `work_limit`, která omezuje pracovní oblast pro osu robotu 1 až -90 a +90 stupňů a externí osu `extax.eax_a` na -1000 mm během vykonávání programu a ručního posuvu (jogging). Proměnná `joint_space` datového typu `shapedata` se používá pro přenos dat z instrukce `WZLimJointDef` do instrukce `WZLimSup`.

Argumenty

```
WZLimJointDef [\Inside] | [\Outside] Shape LowJointVal HighJointVal
```

`[\Inside]`

Datový typ: `switch`

Definovat prostor spoje uvnitř `LowJointVal ... HighJointVal`.

`[\Outside]`

Datový typ: `switch`

Definovat prostor spoje mimo `LowJointVal ... HighJointVal` (inverzní prostor spoje).

`Shape`

Datový typ: `shapedata`

Proměnná pro uložení definovaného prostoru spoje (privátní data pro systém).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`LowJointVal`

Datový typ: `jointtarget`

Pozice souřadnic spoje pro dolní limit prostoru spoje k definování. Určuje pro každý robot osy a externí osy (stupně u rotačních os a mm u lineárních os). Určuje v absolutních spojích (nikoliv v ofsetovém souřadném systému `EOffsSet` nebo `EOffsOn` pro externí osy). Hodnota `9E9` u některé osy znamená, že osa by neměla být dohlížena kvůli dolnímu limitu. Neaktivní externí osa také dává `9E9` v době programování.

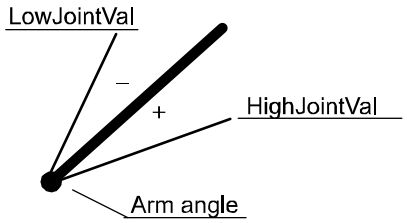
`HighJointVal`

Datový typ: `jointtarget`

Pozice souřadnic spoje pro horní limit prostoru spoje k definování. Určuje pro každý robot osy a externí osy (stupně u rotačních os a mm u lineárních os). Určuje v absolutních spojích (nikoliv v ofsetovém souřadném systému `EOffsSet` nebo `EOffsOn` pro externí osy). Hodnota `9E9` u osy znamená, že osa by neměla být dohlížena kvůli vysokému limitu. Neaktivní externí osa také dává `9E9` v době programování.

`HighJointVal` **minus** `LowJointVal` **pro každou osu musí být větší než 0 pro všechny osy pod dohledem.**

Následující obrázek ukazuje definici prostoru spoje pro rotační osu.



xx0500002281

Následující obrázek ukazuje definici prostoru spoje ru pro lineární osu.

xx0100000002

Vykonávání programu

Definice prostoru spoje je uložena v proměnné typu `shapedata` (argument `Shape`) pro pozdější použití v instrukcích `WZLimSup` nebo `WZDOSet`.

Při používání `WZLimJointDef` společně s `WZDOSet` je digitální výstupní signál nastaven pouze když všechny aktivní osy s dohledem prostoru spoje jsou před nebo uvnitř prostoru spoje.

Při použití `WZLimJointDef` s vnějším prostorem spoje (argument `\Outside`) společně s `WZLimSup` je robot zastaven ihned, jakmile jedna aktivní osa s dohledem prostoru spoje dosáhne prostoru spoje.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.344 WZLimJointDef - Definovat světovou zónu pro omezení ve spojích

World Zones

Pokračování

Při použití `WZLimJointDef` s vnitřním prostorem spoje (argument `\Inside`), společně s `WZLimSup`) je robot zastaven, jakmile poslední aktivní osa s dohledem prostoru spoje dosáhne prostoru spoje. To znamená, že jedna nebo několik os, ale nikoliv všechny aktivní a sledované osy, mohou být uvnitř prostoru spoje ve stejnou dobu.

Při vykonávání instrukce `ActUnit` nebo `DeactUnit` bude status dohledu aktualizován.

Omezení

xx0100000002

POZOR!

Pouze aktivní mechanické jednotky a jejich aktivní osy v době aktivace světové zóny (s instrukcí `WZDOSet` resp. `WZLimSup`) jsou zahrnuty do dohledu pozice HOME resp. do omezení pracovní oblasti. Kromě toho, mechanické jednotky a jejich osy musí být ještě aktivní při pohybu programu nebo ručním posuvu (jogging), aby mohly být sledovány.

Například, jestliže jedna osa s dohledem je mimo svoji HOME pozici spoje, ale je deaktivována, potom to nechrání digitální výstupní signál pro pozici spoje HOME před nastavením, jestliže všechny ostatní aktivní osy s dohledem prostoru spoje jsou uvnitř HOME pozice prostoru spoje. Při nové aktivaci této osy bude zahrnuta do dohledu a systém robotu potom bude mimo HOME pozici spoje a digitální výstup bude resetován.

Syntaxe

```
WZLimJointDef
  [ ['\' Inside] | ['\' Outside] ',']
  [ Shape ':='] <variable (VAR) of shapedata> ','
  [ LowJointVal ':='] <expression (IN) of jointtarget> ','
  [ HighJointVal ':='] <expression (IN) of jointtarget> ';'
```

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Definovat světovou zónu tvaru boxu	*WZBoxDef - Definovat světovou zónu tvaru box na str 1000*
Definovat světovou zónu tvaru válce	*WZCylDef - Definovat světovou zónu tvaru válce na str 1002*
Definovat světovou zónu tvaru koule	*WZSphDef - Definovat světovou zónu tvaru koule na str 1025*
Definovat světovou zónu pro home spoje	*WZHomeJointDef - Definovat světovou zónu pro home spoje na str 1015*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Aktivovat nastavení digitálního výstupu světové zóny	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.345 WZLimSup - Aktivovat dohled limitu světové zóny

Použití

`WZLimSup` (*World Zone Limit Supervision*) se používá pro definování činnosti a aktivaci světové zóny pro dohled nad pracovní oblastí robotu nebo externích os.

Po vykonání instrukce, když TCP robotu dosáhne definované světové zóny nebo když robot/externí osy dosáhnou definované světové zóny ve spojích, pohyb je zastaven jak během vykonávání programu, tak i při ručním posuvu (jogging).

Základní příklady

Následující příklad názorně ukazuje instrukci `WZLimSup`:

Viz také *Další příklady na str 1023*.

Příklad 1

```
VAR wzstationary max_workarea;
...
PROC POWER_ON()
  VAR shapedata volume;
  ...
  WZBoxDef \Outside, volume, corner1, corner2;
  WZLimSup \Stat, max_workarea, volume;
ENDPROC
```

Definice a aktivace stacionární světové zóny `max_workarea` s tvarem oblasti mimo box (dočasně uloženo ve `volume`) a činnost dohledu pracovní oblasti. Robot se zastaví s chybovou zprávou před vstoupením do oblasti mimo box.

Argumenty

```
WZLimSup [\Temp] | [\Stat] WorldZone Shape
```

`[\Temp]`

Temporary

Datový typ: `switch`

Světová zóna k definování je dočasná světová zóna.

`[\Stat]`

Stationary

Datový typ: `switch`

Světová zóna k definování je stacionární světová zóna.

Musí být určen jeden z argumentů `\Temp` nebo `\Stat`.

`WorldZone`

Datový typ: `wztemporary` nebo `wzstationary`

Proměnná nebo perzistentní proměnná, která bude aktualizována s identitou (numerická hodnota) světové zóny.

Při používání přepínače `\Temp`, musí být datový typ `wztemporary`. Při používání přepínače `\Stat`, datový typ musí být `wzstationary`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Shape

Datový typ: `shapedata`

Proměnná, která definuje objem světové zóny.

Vykonávání programu

Definovaná světová zóna je aktivována. Od tohoto momentu je TCP robotu nebo pozice spoje robotu/externích os pod dohledem. Jestliže dosáhne definované oblasti, pohyb je zastaven.

Při použití `WZLimJointDef` **nebo** `WZHomeJointDef`**s vnějším prostorem spoje (argument** `\Outside`**) společně s** `WZLimSup` **je robot zastaven ihned, jakmile jedna aktivní osa s dohledem prostoru spoje dosáhne prostoru spoje.**

Při použití `WZLimJointDef` **nebo** `WZHomeJointDef`**s vnitřním prostorem spoje (argument** `\Inside`**), společně s** `WZLimSup`**) je robot zastaven, jakmile poslední aktivní osa s dohledem prostoru spoje dosáhne prostoru spoje. To znamená, že jedna nebo několik os, ale nikoliv všechny aktivní a sledované osy, mohou být uvnitř prostoru spoje ve stejnou dobu.**

Při vykonávání instrukce `ActUnit` **nebo** `DeactUnit` **bude status dohledu aktualizován.**

Další příklady

Více příkladů jak používat instrukci `WZLimSup` **je názorně uvedeno dole.**

Příklad 1

```
VAR wzstationary box1_invers;
VAR wzstationary box2;

PROC wzone_power_on()
  VAR shapedata volume;
  CONST pos box1_c1:=[500,-500,0];
  CONST pos box1_c2:=[-500,500,500];
  CONST pos box2_c1:=[500,-500,0];
  CONST pos box2_c2:=[200,-200,300];
  ...
  WZBoxDef \Outside, volume, box1_c1, box1_c2;
  WZLimSup \Stat, box1_invers, volume;
  WZBoxDef \Inside, volume, box2_c1, box2_c2;
  WZLimSup \Stat, box2, volume;
ENDPROC
```

Omezení pracovní oblasti pro robot s následujícími stacionárními světovými zónami:

- **Vnější pracovní oblast, když je mimo** `box1_invers`
- **Vnější pracovní oblast, když je uvnitř** `box2`

Jestliže tato rutina je připojena k systémové události POWER ON, potom tyto světové zóny budou vždy aktivní v systému, jak pro pohyby programu, tak i pro ruční posuv (jogging).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.345 WZLimSup - Aktivovat dohled limitu světové zóny

World Zones
Pokračování

Omezení

Světová zóna nemůže být znovu definována pomocí stejné proměnné v argumentu `WorldZone`.

Stacionární světová zóna nemůže být deaktivována, znovu aktivována nebo vymazána v programu RAPID.

Dočasná světová zóna může být deaktivována (`WZDisable`), znovu aktivována (`WZEnable`) nebo vymazána (`WZFree`)) v programu RAPID.

Syntaxe

```
WZLimSup
  [ ['\' Temp] | ['\Stat] ',']
  [ WorldZone ':='] <variable or persistent (INOUT) of wztemporary>
       ','
  [ Shape ':='] <variable (VAR) of shapedata> ';'
```

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Dočasná světová zóna	*wztemporary - Data dočasné světové zóny na str 1640*
Stacionární světová zóna	*wzstationary - Data stacionární světové zóny na str 1638*
Definovat světovou zónu tvaru rovného boxu	*WZBoxDef - Definovat světovou zónu tvaru box na str 1000*
Definovat světovou zónu tvaru koule	*WZSphDef - Definovat světovou zónu tvaru koule na str 1025*
Definovat světovou zónu tvaru válce	*WZCylDef - Definovat světovou zónu tvaru válce na str 1002*
Definovat světovou zónu pro home spoje	*WZHomeJointDef - Definovat světovou zónu pro home spoje na str 1015*
Definovat světovou zónu pro limit spoje	*WZLimJointDef - Definovat světovou zónu pro omezení ve spojích na str 1018*
Aktivovat nastavení digitálního výstupu světové zóny	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.346 WZSphDef - Definovat světovou zónu tvaru koule

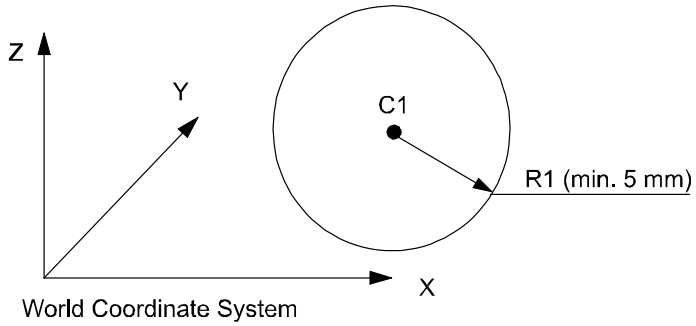
Použití

`WZSphDef` (*World Zone Sphere Definition*) se používá pro definování světové zóny, která má tvar koule.

Základní příklady

Následující příklad názorně ukazuje instrukci `WZSphDef`:

Příklad 1



xx0500002207

```
VAR shapedata volume;
CONST pos C1:=[300,300,200];
CONST num R1:=200;
...
WZSphDef \Inside, volume, C1, R1;
```

Definujte kouli pojmenovanou `volume` podle jejího středu `C1` a jejího poloměru `R1`.

Argumenty

```
WZSphDef [\Inside] | [\Outside] Shape CentrePoint Radius
```

`[\Inside]`

Datový typ: `switch`

Definovat objem uvnitř koule.

`[\Outside]`

Datový typ: `switch`

Definovat objem vně koule (inverzní objem).

Musí být určen jeden z argumentů `\Inside` nebo `\Outside`.

`Shape`

Datový typ: `shapedata`

Proměnná pro uložení definovaného objemu (privátní data pro systém).

`CentrePoint`

Datový typ: `pos`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

1.346 WZSphDef - Definovat světovou zónu tvaru koule

World Zones

Pokračování

Pozice (x,y,z) v mm definující střed koule.

`Radius`

Datový typ: `num`

Poloměr koule v mm.

Vykonávání programu

Definice koule je uložena v proměnné typu `shapedata` (argument `Shape`) pro pozdější použití v instrukcích `WZLimSup` nebo `WZDOSet`.

Omezení

Jestliže robot je použit pro zdůraznění `CentrePoint`, potom musí být pracovní objekt `wobj0` aktivní (použijte komponent `trans` v `robtarget` např. `p1.trans` jako argument).

Syntaxe

```
WZSphDef
  [ ['\' Inside] | ['\' Outside] ',']
  [ Shape ':=']<variable (VAR) of shapedata> ','
  [ CentrePoint ':='] <expression (IN) of pos> ','
  [ Radius ':='] <expression (IN) of num> ';'
```

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Definovat světovou zónu tvaru boxu	*WZBoxDef - Definovat světovou zónu tvaru box na str 1000*
Definovat světovou zónu tvaru válce	*WZCylDef - Definovat světovou zónu tvaru válce na str 1002*
Definovat světovou zónu pro home spoje	*WZHomeJointDef - Definovat světovou zónu pro home spoje na str 1015*
Definovat světovou zónu pro limit spoje	*WZLimJointDef - Definovat světovou zónu pro omezení ve spojích na str 1018*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Aktivovat nastavení digitálního výstupu světové zóny	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2 Funkce

2.1 Abs - Získává absolutní hodnotu

Použití

`Abs` se používá pro získání absolutní hodnoty, to znamená kladné hodnoty numerických dat.

Základní příklady

Následující příklad názorně ukazuje funkci`Abs`.

Viz také *Další příklady na str 1027*.

Příklad 1

```
reg1 := Abs(reg2);
```

`Reg1` má přidělenu absolutní hodnotu `reg2`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Absolutní hodnota, tj. kladná numerická hodnota, například:

Vstupní hodnota	Vrácená hodnota
3	3
-3	3
-2,53	2,53

Argumenty

```
Abs (Value)
```

`Value`

Datový typ: `num`

Vstupní hodnota.

Další příklady

Více příkladů funkce `Abs` je názorně uvedeno dole.

Příklad 1

```
TPReadNum no_of_parts, "How many parts should be produced? ";
no_of_parts := Abs(no_of_parts);
```

Operátor je požádán, aby zadal počet kusů, které mají být vyrobeny. Aby bylo zajištěno, že hodnota je větší než nula, hodnota zadaná operátorem je nastavena kladně.

Syntaxe

```
Abs '('
  [ Value ':=' ] < expression (IN) of num >')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.1 Abs - Získává absolutní hodnotu

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.2 AbsDnum - Získává absolutní hodnotu dnum

Použití

`AbsDnum` se používá pro získání absolutní hodnoty, to znamená kladné hodnoty numerické hodnoty `dnum`

Základní příklady

Následující příklad názorně ukazuje funkci`AbsDnum`.

Viz také *Další příklady na str 1029*.

Příklad 1

```
VAR dnum value1;
VAR dnum value2:=-20000000;
value1 := AbsDnum(value2);
```

`Value1` má přidělenu absolutní hodnotu `value2`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Absolutní hodnota, tj. kladná numerická hodnota, například:

Vstupní hodnota	Vrácená hodnota
3	3
-3	-3
-2,53	2,53
-4503599627370496	4503599627370496

Argumenty

```
AbsDnum (Value)
```

`Value`

Datový typ: `dnum`

Vstupní hodnota.

Další příklady

Více příkladů funkce `AbsDnum` je názorně uvedeno dole.

Příklad 1

```
TPReadDnum no_of_parts, "How many parts should be produced? ";
no_of_parts := AbsDnum(no_of_parts);
```

Operátor je požádán, aby zadal počet kusů, které mají být vyrobeny. Aby bylo zajištěno, že hodnota je větší než nula, hodnota zadaná operátorem je nastavena kladně.

Syntaxe

```
AbsDnum '('
  [ Value ':=' ] < expression (IN) of dnum > ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.2 AbsDnum - Získává absolutní hodnotu dnum

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.3 ACos - Vypočítá hodnotu arkuskosinu

Použití

ACos (*Arc Cosine*) se používá k výpočtu hodnoty arkuskosinu na datových typech num.

Základní příklady

Následující příklad názorně ukazuje funkci ACos.

Příklad 1

```
VAR num angle;
VAR num value;
...
...
angle := ACos(value);
```

angle získá hodnotu arkuskosinu value.

Vratná hodnota (Vrátit hodnotu)

Datový typ: num

Hodnota arkuskosinu ve stupních, rozsah [0, 180].

Argumenty

```
ACos (Value)
```

Value

Datový typ: num

Hodnota argumentu musí být v rozsahu [-1, 1].

Omezení

Vykonání funkce Acos(x) vykáže chybu, jestliže x je mimo rozsah [-1, 1].

Syntaxe

```
Acos '('
  [Value ':='] <expression (IN) of num>')'
```

Funkce s vrácenou hodnotou datového typu num.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.4 ACosDnum - Vypočítá hodnotu arkuskosinu

Použití

ACosDnum (*Arc Cosine dnum*) se používá k výpočtu hodnoty arkuskosinu na datových typech dnum.

Základní příklady

Následující příklad názorně ukazuje funkciACosDnum.

Příklad 1

```
VAR dnum angle;
VAR dnum value;
...
...
angle := ACosDnum(value);
```

angle získá hodnotu arkuskosinu value.

Vratná hodnota (Vrátit hodnotu)

Datový typ: dnum

Hodnota arkuskosinu ve stupních, rozsah [0, 180].

Argumenty

```
ACosDnum (Value)
```

Value

Datový typ: dnum

Hodnota argumentu musí být v rozsahu [-1, 1].

Omezení

Vykonání funkce AcosDnum(x) vykáže chybu, jestliže x je mimo rozsah [-1, 1].

Syntaxe

```
AcosDnum '('
  [Value ':='] <expression (IN) of dnum>')'
```

Funkce s vrácenou hodnotou datového typu dnum.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.5 AInput - Přečte hodnotu analogového vstupního signálu

Použití

`AInput` se používá pro čtení aktuální hodnoty analogového vstupního signálu.

POZNÁMKA

Všimněte si, že funkce `AInput` je zděděná funkce, kterou již není nutné používat. Viz příklady alternativních a doporučených způsobů programování.

Základní příklady

Následující příklad názorně ukazuje funkci`AInput`.

Viz také *Další příklady na str 1034*.

Příklad 1

```
IF AInput(ai1) < 1.5 THEN ...
...
IF ai1 < 1.5 THEN ...
```

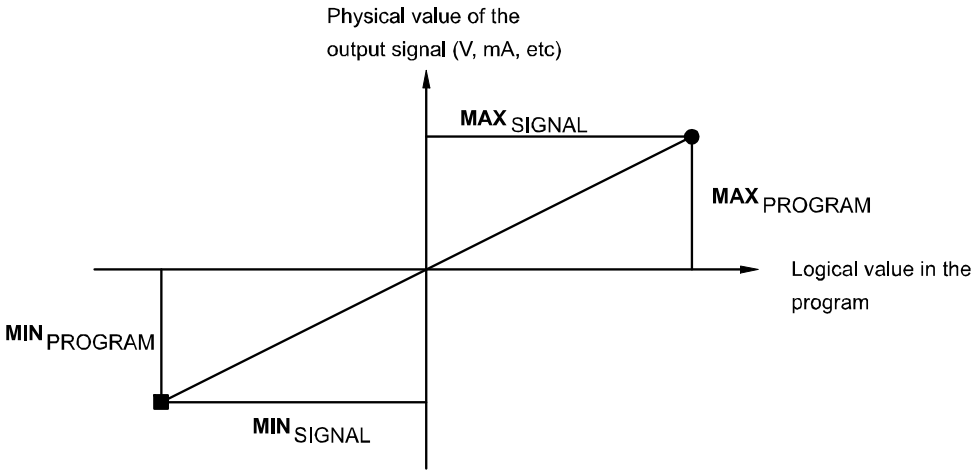
Jestliže aktuální hodnota signálu `ai1` je menší než 1,5, potom ...

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Aktuální hodnota signálu.

Aktuální hodnota je škálována (v souladu se systémovými parametry) předtím, než je přečtena programem RAPID. Na následujícím obrázku je schéma, jak jsou škálovány hodnoty analogových signálů.



xx0500002408

Argumenty

```
AInput (Signal)
```

`Signal`

Datový typ: `signalai`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.5 AInput - Přečte hodnotu analogového vstupního signálu
Pokračování

Jméno analogového vstupu, které bude přečteno.

Další příklady

Více příkladů jak používat instrukci `AInput` je názorně uvedeno dole.

Příklad 1

```
WHILE AInput(current) > 35 DO ...
...
WHILE current > 35 DO ...
```

Dokud je aktuální hodnota signálu `current` větší než 35, proveďte ...

Příklad 2

```
deviation := 3 * AInput(sensor) + 10;
...
deviation := 3 * sensor + 10;
```

Odchylka se vypočítá na základě hodnoty signálu `sensor` a uloží se do proměnné `deviation`.

Syntaxe

```
AInput '('
  [ Signal ':=' ] < variable (VAR) of signalai > ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.6 AND - Vyhodnocuje logickou hodnotu

Použití

AND je funkce, která se používá k vyhodnocení dvou podmíněných výrazů (true/false).

Základní příklady

Následující příklady názorně ukazují funkciAND.

Příklad 1

```
VAR num a;
VAR num b;
VAR bool c;
...
c := a>5 AND b=3;
```

Vrácená hodnota c je TRUE, jestliže a je větší než 5 a b je rovno 3. Jinak je vrácená hodnota FALSE.

Příklad 2

```
VAR num mynum;
VAR string mystring;
VAR bool mybool;
VAR bool result;
...
result := mystring="Hello" AND mynum<15 OR mybool;
```

Vrácená hodnota result je TRUE, jestliže oba mystring "Hello" a mynum jsou menší než 15. Nebo jestliže mybool je TRUE. Jinak je vrácená hodnota FALSE.

Příkaz AND je vyhodnocen jako první, potom příkaz OR. Toto je ilustrováno závorkami v řádce dole.

```
result := (mystring="Hello" AND mynum<15) OR mybool;
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: bool

Vrácená hodnota je TRUE, jestliže oba podmíněné výrazy jsou správné, jinak je vrácená hodnota FALSE.

Syntaxe

```
<expression of bool> AND <expression of bool>
```

Funkce s vrácenou hodnotou datového typu bool.

Související informace

Pro informace o	Viz
Logická bitová AND - operace na datech byte	*BitAnd - Logická bitová AND - operace na datech byte na str 1048*
Logická bitová AND - operace na datech dnum	*BitAndDnum - Logická bitová AND - operace na datech dnum na str 1050*
OR	*OR - Vyhodnocuje logickou hodnotu na str 1242*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.6 AND - Vyhodnocuje logickou hodnotu

Pokračování

Pro informace o	Viz
XOR	*XOR - Vyhodnocuje logickou hodnotu na str 1435*
NOT	*NOT - Invertuje logickou hodnotu na str 1233*
Výrazy	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.7 AOutput - Čte hodnotu analogového výstupního signálu

Použití

`AOutput` se používá pro čtení aktuální hodnoty analogového výstupního signálu.

Základní příklady

Následující příklad názorně ukazuje funkci`AOutput`.

Příklad 1

```
IF AOutput(ao4) > 5 THEN ...
```

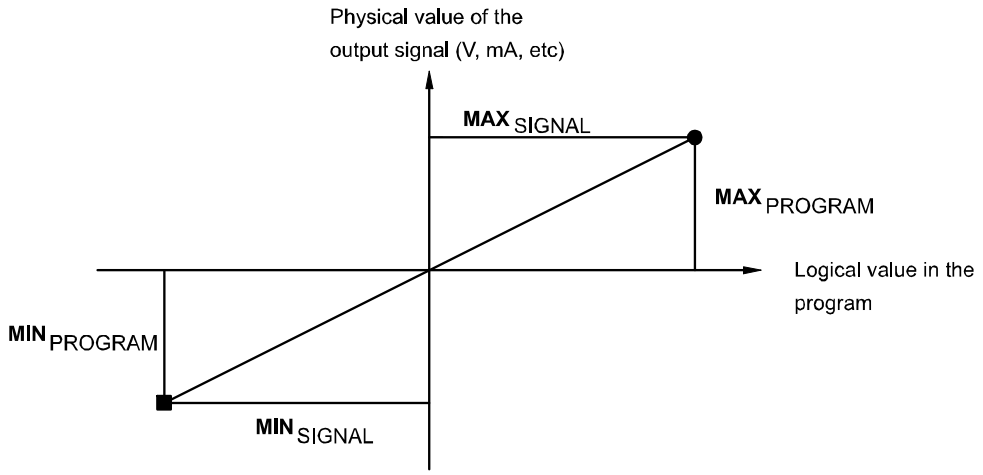
Jestliže aktuální hodnota signálu `ao4` je větší než `5`, potom `...`

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Aktuální hodnota signálu.

Aktuální hodnota je škálována (v souladu se systémovými parametry) předtím, než je přečtena programem RAPID. Na následujícím obrázku je schéma, jak jsou škálovány hodnoty analogových signálů.



xx0500002408

Argumenty

```
AOutput (Signal)
```

`Signal`

Datový typ: `signalao`

Jméno analogového výstupu, které bude přečteno.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.7 AOutput - Čte hodnotu analogového výstupního signálu

RobotWare - OS

Pokračování

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
AOutput '('
  [ Signal ':=' ] < variable (VAR) of signalao > ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Nastavit analogový výstupní signál	*SetAO - Mění hodnotu analogového výstupního signálu na str 620*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.8 ArgName - Získává jméno argumentu

Použití

`ArgName` (*Argument Name*) se používá k získání jména původního datového objektu pro aktuální argument nebo aktuální data.

Základní příklady

Následující příklad názorně ukazuje funkci`ArgName`.

Viz také *Další příklady na str 1040*.

Příklad 1

```
VAR num chales :=5;
...
proc1 chales;
PROC proc1 (num par1)
  VAR string name;
  ...
  name:=ArgName(par1);
  TPWrite "Argument name "+name+" with value "\Num:=par1;
ENDPROC
```

Proměnné `name` je přidělena řetězcová hodnota "`chales`" a na FlexPendantu je zapsán následující řetězec: "Argument name chales with value 5".

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Jméno objektu původních dat.

Argumenty

```
ArgName (Parameter [\ErrorNumber])
```

`Parameter`

Datový typ: `anytype`

Formální identifikátor parametru (pro rutinu, ve které je umístěno `ArgName`) nebo identita dat.

Všechny typy dat se strukturou jako je atomická, záznam, komponent záznamu, pole nebo prvek pole by mohly být použity.

`ErrorNumber`

Datový typ: `errnum`

Proměnná (před použitím je nastavena systémem na 0), která bude držet chybový kód, když argument je hodnota výrazu, argument není přítomen nebo argument je typu přepínač. Jestliže je tato volitelná proměnná vypuštěna, potom bude vykonán chybový handler.

Vykonávání programu

Funkce vrací jméno původního datového objektu pro celý objekt typu konstanta, proměnná nebo perzistent. Původní datový objekt může být globální, lokální v programovém modulu nebo lokální v rutině (normální rámcová pravidla RAPID).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže je součástí datového objektu, potom je jméno celého datového objektu vráceno.

Další příklady

Více příkladů funkce `ArgName` je názorně uvedeno dole.

Převést z celého čísla na řetězec

Tuto funkci je také možné používat k převodu z identifikátoru na `string` určením identifikátoru v argumentu `Parameter` pro jakýkoliv datový objekt s rámcem globální, lokální v modulu nebo lokální v rutině:

```
VAR num chales :=5;
...
proc1;

PROC proc1 ()
  VAR string name;
  ...
  name:=ArgName(chales);
  TPWrite "Global data object "+name+" has value "\Num:=chales;
ENDPROC
```

Proměnné `name` je přidělena řetězcová hodnota "`chales`" a na FlexPendantu je zapsán následující řetězec: "Global data object chales has value 5".

Volání rutiny v několika krocích

Všimněte si, že funkce vrací jméno **původního** datového objektu:

```
VAR num chales :=5;
...
proc1 chales;
...
PROC proc1 (num parameter1)
  ...
  proc2 parameter1;
  ...
ENDPROC

PROC proc2 (num par1)
  VAR string name;
  ...
  name:=ArgName(par1);
  TPWrite "Original data object name "+name+" with value"
       \Num:=par1;
ENDPROC
```

Proměnné `name` je přidělena řetězcová hodnota "`chales`" a na FlexPendantu je zapsán následující řetězec: "Original data object chales with value 5".

Potlačit vykonávání v chybovém handleru

```
PROC main()
  VAR string mystring:="DUMMY";
  proc1 mystring;
  proc1 "This is a test";
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  ...
ENDPROC

PROC proc1 (string par1)
  VAR string name;
  VAR errnum myerrnum;

  name := ArgName(par1 \ErrorNumber:=myerrnum);
  IF myerrnum=ERR_ARGNAME THEN
    TPWrite "The argument par1 is an expression value";
    TPWrite "The name of the argument can not be evaluated";
  ELSE
    TPWrite "The name on the argument is "+name;
  ENDIF
ENDPROC
```

Proměnné `name` je přidělena řetězcová hodnota "`mystring`", když první volání k `proc1` je provedeno. Když je provedeno druhé volání k `proc1`, prázdný řetězec je přidělen ke jménu. Na FlexPendantu je zapsán následující řetězec: "`The argument par1 is an expression value`" a "`The name of the argument can not be evaluated`".

Řešení chyb

Jestliže se objeví jedna z následujících chyb, potom je systémová proměnná `ERRNO` nastavena na `ERR_ARGNAME`:

- Argument je hodnota výrazu
- Argument není přítomen
- Argument je typu přepínač

Tato chyba může být potom zpracována v chybovém handleru.

Syntaxe

```
ArgName '('
  [ Parameter ':=' ] < reference (REF) of any type>
  ['\' ErrorNumber ':=' <var or pers (INOUT) of errnum>] ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Advanced RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.9 ASin - Vypočítá hodnotu arkussinu

Použití

`ASin` (***Arc Sine***) se používá k výpočtu hodnoty arkuskosinu na datových typech `num`.

Základní příklady

Následující příklad názorně ukazuje funkci`ASin`

Příklad 1

```
VAR num angle;
VAR num value;
...
...
angle := ASin(value);
```

`angle` získá hodnotu arkuskosinu `value`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Hodnota arkuskosinu vyjádřená ve stupních, rozsah [-90, 90].

Argumenty

```
ASin (Value)
```

`Value`

Datový typ: `num`

Hodnota argumentu musí být v rozsahu [-1, 1].

Omezení

Vykonání funkce `ASin(x)` vykáže chybu, jestliže x je mimo rozsah [1, -1].

Syntaxe

```
ASin '('
  [Value ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.10 ASinDnum - Vypočítá hodnotu arkussinu

Použití

`ASinDnum` (*Arc Sine dnum*) se používá k výpočtu hodnoty arkuskosinu na datových typech `dnum`.

Základní příklady

Následující příklad názorně ukazuje funkci`ASinDnum`

Příklad 1

```
VAR dnum angle;
VAR dnum value;
...
...
angle := ASinDnum(value);
```

`angle` získá hodnotu arkuskosinu `value`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Hodnota arkuskosinu vyjádřená ve stupních, rozsah [-90, 90].

Argumenty

```
ASinDnum (Value)
```

`Value`

Datový typ: `dnum`

Hodnota argumentu musí být v rozsahu [-1, 1].

Omezení

Vykonání funkce `ASinDnum(x)` vykáže chybu, jestliže x je mimo rozsah [1, -1].

Syntaxe

```
ASinDnum '('
  [Value ':='] <expression (IN) of dnum> ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.11 ATan - Vypočítá hodnotu arkustangens

Použití

ATan (*Arc Tangent*) se používá k výpočtu hodnoty arkustangens na datových typech num.

Základní příklady

Následující příklad názorně ukazuje funkciATan.

Příklad 1

```
VAR num angle;
VAR num value;
...
...
angle := ATan(value);
```

angle získá hodnotu arkustangens value.

Vratná hodnota (Vrátit hodnotu)

Datový typ: num

Hodnota arkustangens vyjádřená ve stupních, rozsah [-90, 90].

Argumenty

```
ATan (Value)
```

Value

Datový typ: num

Hodnota argumentu.

Syntaxe

```
ATan '('
  [Value ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu num.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Arkustangens s vrácenou hodnotou v rozsahu [-180, 180]	*ATan2 - Vypočítá hodnotu arkustangens2 na str 1046*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.12 ATanDnum - Vypočítá hodnotu arkustangens

Použití

`ATanDnum` (*Arc Tangent dnum*) se používá k výpočtu hodnoty arkustangens na datových typech `dnum`.

Základní příklady

Následující příklad názorně ukazuje funkci`ATanDnum`.

Příklad 1

```
VAR dnum angle;
VAR dnum value;
...
...
angle := ATanDnum(value);
```

`angle` získá hodnotu arkustangens `value`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Hodnota arkustangens vyjádřená ve stupních, rozsah [-90, 90].

Argumenty

```
ATanDnum (Value)
```

`Value`

Datový typ: `dnum`

Hodnota argumentu.

Syntaxe

```
ATanDnum '('
  [Value ':='] <expression (IN) of dnum> ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Arkustangens s vrácenou hodnotou v rozsahu [-180, 180]	*ATan2 - Vypočítá hodnotu arkustangens2 na str 1046*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.13 ATan2 - Vypočítá hodnotu arkustangens2

Použití

ATan2 (*Arc Tangent2*) se používá k výpočtu hodnoty arkustangens2 na datových typech num.

Základní příklady

Následující příklad názorně ukazuje funkciATan2.

Příklad 1

```
VAR num angle;
VAR num x_value;
VAR num y_value;
...
...
angle := ATan2(y_value, x_value);
```

angle získá hodnotu arkustangens y_value/x_value.

Vratná hodnota (Vrátit hodnotu)

Datový typ: num

Hodnota arkustanges vyjádřená ve stupních, rozsah [-180, 180]. Hodnota bude rovna ATan(y/x), ale v rozsahu [-180, 180], jelikož funkce používá znaménko obou argumentů k určení kvadrantu vrácené hodnoty.

Argumenty

```
ATan2 (Y X)
```

Y

Datový typ: num

Hodnota argumentu činitele.

X

Datový typ: num

Hodnota argumentu jmenovatele.

Syntaxe

```
ATan2 '('
  [Y ':='] <expression (IN) of num> ','
  [X ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu num.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Arkutangens pouze s jedním argumentem	*ATan - Vypočítá hodnotu arkustangens na str 1044*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.14 ATan2Dnum - Vypočítá hodnotu arkustangens2

Použití

`ATan2Dnum` (*Arc Tangent2 dnum*) se používá k výpočtu hodnoty arkustangens2 na datových typech `dnum`.

Základní příklady

Následující příklad názorně ukazuje funkci`ATan2Dnum`.

Příklad 1

```
VAR dnum angle;
VAR dnum x_value;
VAR dnum y_value;
...
...
angle := ATan2Dnum(y_value, x_value);
```

`angle` získá hodnotu arkustangens `y_value/x_value`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Hodnota arkustanges vyjádřená ve stupních, rozsah [-180, 180]. Hodnota bude rovna ATanDnum(y/x), ale v rozsahu [-180, 180], jelikož funkce používá znaménko obou argumentů k určení kvadrantu vrácené hodnoty.

Argumenty

```
ATan2Dnum (Y X)
```

Y

Datový typ: `dnum`

Hodnota argumentu činitele.

X

Datový typ: `dnum`

Hodnota argumentu jmenovatele.

Syntaxe

```
ATan2Dnum '('
  [Y ':='] <expression (IN) of dnum> ','
  [X ':='] <expression (IN) of dnum> ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Arkutangens pouze s jedním argumentem	*ATan - Vypočítá hodnotu arkustangens na str 1044*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.15 BitAnd - Logická bitová AND - operace na datech byte

Použití

`BitAnd` se používá k vykonání logické bitové operace `AND` na datových typech byte.

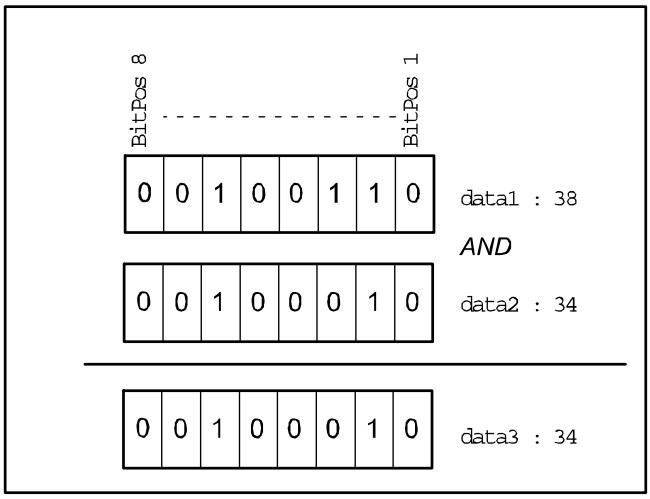
Základní příklady

Následující příklad názorně ukazuje funkci`BitAnd`.

Příklad 1

```
VAR byte data1 := 38;
VAR byte data2 := 34;
VAR byte data3;

data3 := BitAnd(data1, data2);
```

Logická bitová AND - operace (viz následující obrázek) je vykonána na `data1` a `data2`. Výsledek je vrácen do `data3` (reprezentace celého čísla).



xx0500002454

Vratná hodnota (Vrátit hodnotu)

Datový typ: `byte`

Výsledek logické bitové operace `AND` v reprezentaci celého čísla.

Argumenty

```
BitAnd (BitData1 BitData2)
```

`BitData1`

Datový typ: `byte`

Bitová data 1 v reprezentaci celého čísla.

`BitData2`

Datový typ: `byte`

Bitová data 2 v reprezentaci celého čísla.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Rozsah pro datový typ je `byte` je 0 - 255.

Syntaxe

```
BitAnd '('
  [BitData1 ':='] <expression (IN) of byte> ','
  [BitData2 ':='] <expression (IN) of byte> ')'
```

Funkce s vrácenou hodnotou datového typu `byte`.

Související informace

Pro informace o	Viz
Logická bitová AND - operace na datech byte	*BitOr - Logická bitová OR- operace na datech byte na str 1065*
Logická bitová XOR - operace na datech byte	*BitXOr - Logická bitová XOR operace na datech byte na str 1073*
Logická bitová NEGATION - operace na datech byte	*BitNeg - Logická bitová NEGATION - operace na datech byte na str 1061*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.16 BitAndDnum - Logická bitová AND - operace na datech dnum

Použití

`BitAndDnum` se používá k vykonání logické bitové operace `AND` na datových typech `dnum`.

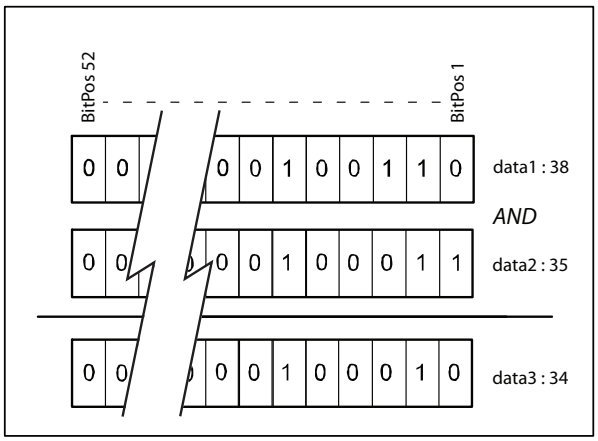
Základní příklady

Následující příklad názorně ukazuje funkci`BitAndDnum`.

Příklad 1

```
VAR dnum data1 := 38;
VAR dnum data2 := 35;
VAR dnum data3;

data3 := BitAndDnum(data1, data2);
```

Logická bitová AND - operace (viz následující obrázek) bude vykonána na `data1` a `data2`. Výsledek je vrácen do `data3` (reprezentace celého čísla).



xx1200000007

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Výsledek logické bitové operace `AND` v reprezentaci celého čísla.

Argumenty

```
BitAndDnum (Value1 Value2)
```

`Value1`

Datový typ: `dnum`

Datová hodnota prvního bitu v reprezentaci celého čísla.

`Value2`

Datový typ: `dnum`

Datová hodnota druhého bitu v reprezentaci celého čísla.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Rozsah pro datový typ je `dnum` je 0 - 4503599627370495.

Syntaxe

```
BitAndDnum '('
  [Value1 ':='] <expression (IN) of dnum> ','
  [Value2 ':='] <expression (IN) of dnum> ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Logická bitová AND - operace na datech `byte`	*BitAnd - Logická bitová AND - operace na datech byte na str 1048*
Datový typ `dnum`	*dnum - Dvojité numerické hodnoty na str 1485*
Logická bitová OR - operace na datech `dnum`	*BitOrDnum - Logická bitová OR operace na datech dnum na str 1067*
Logická bitová XOR - operace na datech `dnum`	*BitXOrDnum - Logická bitová XOR operace na datech dnum na str 1075*
Logická bitová NEGATION - operace na datech `dnum`	*BitNegDnum - Logická bitová NEGATION operace na datech dnum na str 1063*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.17 BitCheck - Zkontrolujte, jestli je nastaven určený bit v datech byte

Použití

`BitCheck` se používá ke kontrole, jestli určený bit v definovaných `byte` datech je nastaven na 1.

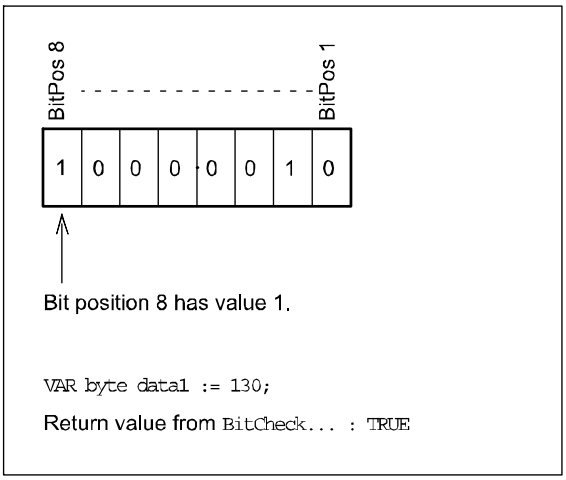
Základní příklady

Následující příklad názorně ukazuje funkci`BitCheck`.

Příklad 1

```
CONST num parity_bit := 8;
VAR byte data1 := 130;

IF BitCheck(data1, parity_bit) = TRUE THEN
  ...
ELSE
  ...
ENDIF
```

Bit číslo `8` (`parity_bit`) v proměnné `data1` je kontrolován, například, jestliže určený bit je nastaven na 1 v proměnné `data1`, potom se tato funkce vrátí na `TRUE`. Bitová kontrola datového typu `byte` je uvedena na následujícím obrázku.



xx0500002442

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže určený bit je nastaven na 1, `FALSE`, jestliže určený bit je nastaven na 0.

Argumenty

```
BitCheck (BitData BitPos)
```

`BitData`

Datový typ: `byte`

Bitová data, v zastoupení celého čísla, která budou zkontrolována.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`BitPos`

Bit Position

Datový typ: `num`

Pozice bitu (1-8) v `BitData` ke kontrole.

Omezení

Rozsah pro datový typ je `byte` 0 - 255 desítkový.

Pozice bitu je platná od 1 - 8.

Syntaxe

```
BitCheck '('
  [BitData ':='] <expression (IN) of byte> ','
  [BitPos ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Nastavit určený bit v datech byte	*BitSet - Nastavit určený bit do dat byte nebo dnum na str 38*
Vyčistit určený bit v datech byte	*BitClear - Vyčistit určený bit v bytu nebo dnum datech na str 35*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.18 BitCheckDnum - Zkontrolujte, jestli je nastaven určený bit v datech dnum

Použití

`BitCheckDnum` se používá ke kontrole, jestli určený bit v definovaných `dnum` datech je nastaven na 1.

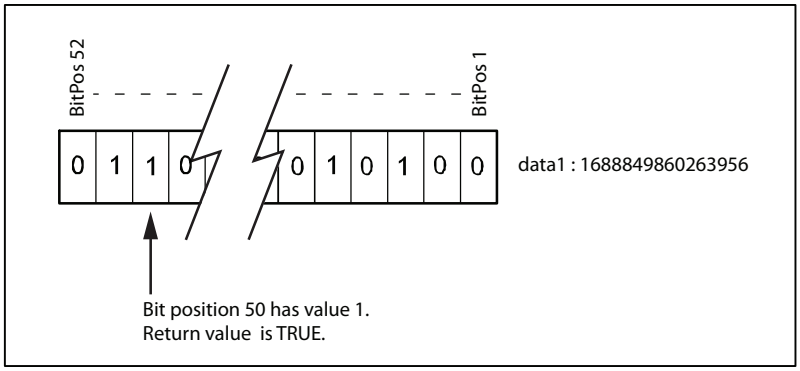
Základní příklady

Následující příklad názorně ukazuje funkci`BitCheckDnum`.

Příklad 1

```
CONST num check_bit := 50;
VAR dnum data1 := 1688849860263956;

IF BitCheckDnum(data1, check_bit) = TRUE THEN
  ...
ELSE
  ...
ENDIF
```

Bit číslo `50` (`check_bit`) v proměnné `data1` bude zkontrolován, například, jestliže určený bit je 1 v proměnné `data1`, potom se tato funkce vrátí na `TRUE`. Bitová kontrola datového typu `dnum` je uvedena na obrázku dole.



xx1200000016

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže určený bit je nastaven na 1, `FALSE`, jestliže určený bit je nastaven na 0.

Argumenty

```
BitCheckDnum (Value BitPos)
```

`Value`

Datový typ: `dnum`

Bitová data, v zastoupení celého čísla, která budou zkontrolována.

`BitPos`

Bit Position

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Datový typ: `num`

Pozice bitu (1-52) v `Value` ke kontrole.

Omezení

Rozsah pro datový typ je `dnum` 0 - 4503599627370495 desítkový.

Pozice bitu je platná od 1 - 52.

Syntaxe

```
BitCheckDnum '('
  [Value ':='] <expression (IN) of dnum> ','
  [BitPos ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Zkontrolujte, jestli je nastaven určený bit v datech **byte**	*BitCheck - Zkontrolujte, jestli je nastaven určený bit v datech byte na str 1052*
Datový typ `dnum`	*dnum - Dvojité numerické hodnoty na str 1485*
Nastavit určený bit v datech `byte` nebo `dnum`	*BitSet - Nastavit určený bit do dat byte nebo dnum na str 38*
Vyčistit určený bit v datech `byte` nebo `dnum`	*BitClear - Vyčistit určený bit v bytu nebo dnum datech na str 35*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.19 BitLSh - Logická bitová LEFT SHIFT - operace na bajtech

Použití

`BitLSh` (***Bit Left Shift***) se používá k vykonání logické bitové LEFT SHIFT operace na datových typech `byte`.

Základní příklady

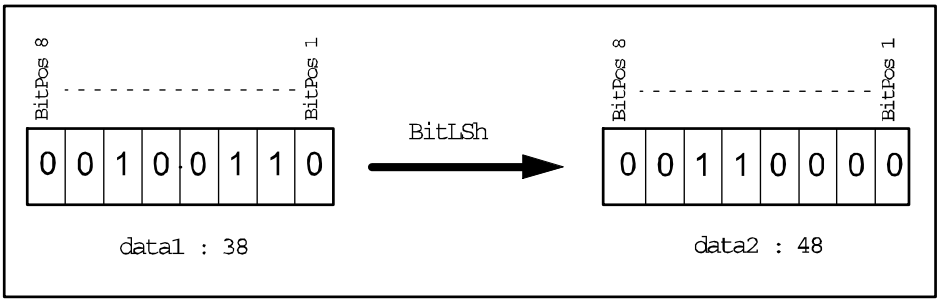
Následující příklad názorně ukazuje funkci`BitLSh`.

Příklad 1

```
VAR num left_shift := 3;
VAR byte data1 := 38;
VAR byte data2;

data2 := BitLSh(data1, left_shift);
```

Logická bitová LEFT SHIFT operace bude vykonána na `data1` s `3` (`left_shift`) kroky levého posunu, a výsledek bude vrácen do `data2` (reprezentace celého čísla).

Následující obrázek ukazuje logickou bitovou LEFT SHIFT operaci.



xx0500002457

Vratná hodnota (Vrátit hodnotu)

Datový typ: `byte`

Výsledek logické bitové LEFT SHIFT operace v reprezentaci celého čísla.

Pravé bitové buňky budou naplněny 0-bity.

Argumenty

```
BitLSh (BitData ShiftSteps)
```

`BitData`

Datový typ: `byte`

Bitová data, v reprezentaci celého čísla, která budou posunuta.

`ShiftSteps`

Datový typ: `num`

Počet logických posunutí (1 - 8), která budou vykonána.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Rozsah pro datový typ je `byte` je 0 - 255.

Argument `ShiftSteps` je platný od 1 - 8 podle jednoho bajtu.

Syntaxe

```
BitLSh '('
  [BitData ':='] <expression (IN) of byte> ','
  [ShiftSteps ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `byte`.

Související informace

Pro informace o	Viz
Logická bitová RIGHT SHIFT operace na datech byte	*BitRSh - Logická bitová RIGHT SHIFT operace na bajtech na str 1069*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Matematika - Bitové funkce*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.20 BitLShDnum - Logická bitová LEFT SHIFT operace na dnum

Použití

`BitLShDnum` (*Bit Left Shift dnum*) se používá k vykonání logické bitové LEFT SHIFT operace na datových typech `dnum`.

Základní příklady

Následující příklad názorně ukazuje funkci`BitLShDnum`.

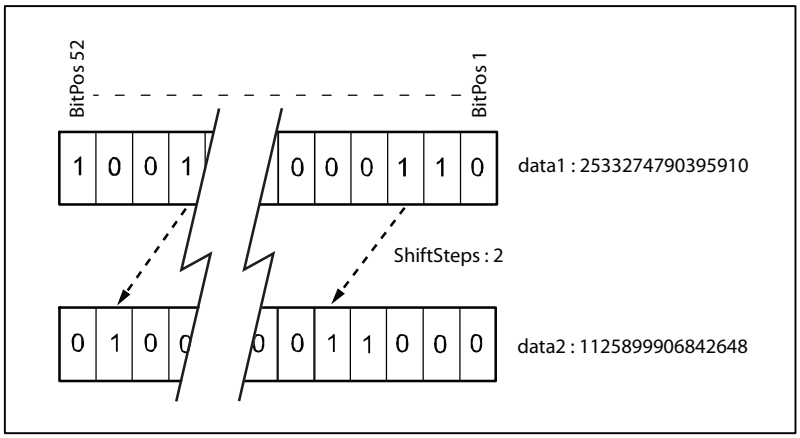
Viz také *Další příklady na str 1059*.

Příklad 1

```
VAR num left_shift := 2;
VAR dnum data1 := 2533274790395910;
VAR dnum data2;

data2 := BitLShDnum(data1, left_shift);
```

Logická bitová LEFT SHIFT operace bude vykonána na `data1` s `2` (`left_shift`) kroky levého posunu, a výsledek bude vrácen do `data2` (reprezentace celého čísla).

Následující obrázek ukazuje logickou bitovou LEFT SHIFT operaci.



xx1200000008

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Výsledek logické bitové LEFT SHIFT operace v reprezentaci celého čísla.

Pravé bitové buňky budou naplněny 0-bity.

Argumenty

```
BitLShDnum (Value ShiftSteps [\Size])
```

`Value`

Datový typ: `dnum`

Bitová data, v reprezentaci celého čísla, která budou posunuta.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`ShiftSteps`

Datový typ: `num`

Počet logických posunutí (1 - 52), která budou vykonána.

`Size`

Datový typ: `num`

Velikost (počet bitů), která by měla být brána v úvahu při provádění logické bitové LEFT SHIFT operace na argumentu `Value`. Velikost je platná od 1 - 52.

Omezení

Rozsah pro datový typ je `dnum` je 0 - 4503599627370495.

Argument `ShiftSteps` je platný od 1 - 52, jelikož jeden `dnum` je 52 bitů.

Další příklady

Více příkladů funkce `BitLshDnum` je názorně uvedeno dole.

Příklad 1

```
VAR dnum result;
VAR dnum data1:=221;
! Only consider the 8 lowest bits
result := BitLshDnum(data1, 4 \Size:=8);
TPWrite "" \Dnum:=result;
! Consider all 52 bits in the dnum datatype
result := BitLshDnum(data1, 4);
TPWrite "" \Dnum:=result;
```

Logická bitová LEFT SHIFT operace bude vykonána na `data1` a výsledek bude vrácen do `result` (reprezentace celého čísla). První hodnota, která bude zapsána na FlexPendant, je 208. Druhá hodnota, která bude zapsána na FlexPendant, je 3536.

Syntaxe

```
BitLShDnum '('
  [Value ':='] <expression (IN) of dnum> ','
  [ShiftSteps ':='] <expression (IN) of num>
  ['\' Size ':=' < expression (IN) of num>]
  ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Logická bitová LEFT SHIFT operace na datech **byte**	*BitLSh - Logická bitová LEFT SHIFT - operace na bajtech na str 1056*
Datový typ `dnum`	*dnum - Dvojité numerické hodnoty na str 1485*
Logická bitová RIGHT SHIFT operace na datech dnum	*BitRShDnum - Logická bitová RIGHT SHIFT operace na dnum na str 1071*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.20 BitLShDnum - Logická bitová LEFT SHIFT operace na dnum

Pokračování

Pro informace o	Viz
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Matematika - Bitové funkce*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.21 BitNeg - Logická bitová NEGATION - operace na datech byte

Použití

`BitNeg` (***Bit Negation***) se používá k vykonání logické bitové operace `NEGATION` (vlastní komplement) na datových typech `byte`.

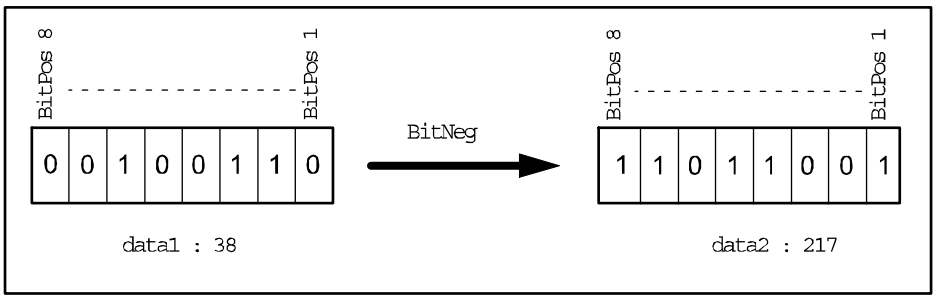
Základní příklady

Následující příklad názorně ukazuje funkci`BitNeg`.

Příklad 1

```
VAR byte data1 := 38;
VAR byte data2;

data2 := BitNeg(data1);
```

Logická bitová operace `NEGATION` (viz obrázek dole) bude vykonána na `data1` a výsledek bude vrácen do `data2` (reprezentace celého čísla).



xx0500002456

Vratná hodnota (Vrátit hodnotu)

Datový typ: `byte`

Výsledek logické bitové operace `NEGATION` v reprezentaci celého čísla.

Argumenty

```
BitNeg (BitData)
```

`BitData`

Datový typ: `byte`

Data byte v reprezentaci celého čísla.

Omezení

Rozsah pro datový typ je `byte` je 0 - 255.

Syntaxe

```
BitNeg '('
  [BitData ':='] <expression (IN) of byte>
  ')'
```

Funkce s vrácenou hodnotou datového typu `byte`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.21 BitNeg - Logická bitová NEGATION - operace na datech byte

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Logická bitová AND operace na datech byte	*BitAnd - Logická bitová AND - operace na datech byte na str 1048*
Logická bitová AND - operace na datech byte	*BitOr - Logická bitová OR- operace na datech byte na str 1065*
Logická bitová XOR - operace na datech byte	*BitXOr - Logická bitová XOR operace na datech byte na str 1073*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.22 BitNegDnum - Logická bitová NEGATION operace na datech dnum

Použití

`BitNegDnum` (*Bit Negation dnum*) se používá k vykonání logické bitové operace `NEGATION` (vlastní komplement) na datových typech `dnum`.

Základní příklady

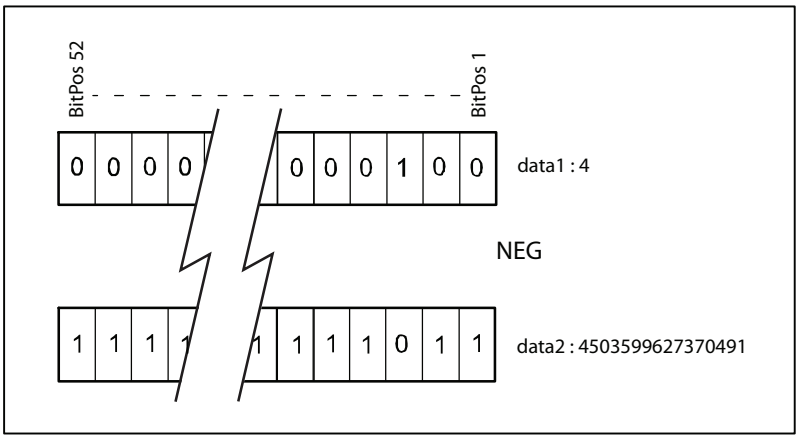
Následující příklad názorně ukazuje funkci`BitNegDnum`.

Viz také *Další příklady na str 1064*.

Příklad 1

```
VAR dnum data1 := 4;
VAR dnum data2;

data2 := BitNegDnum(data1);
```

Logická bitová operace `NEGATION` (viz obrázek dole) bude vykonána na `data1` a výsledek bude vrácen do `data2` (reprezentace celého čísla).



xx1200000012

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Výsledek logické bitové operace `NEGATION` v reprezentaci celého čísla.

Argumenty

```
BitNegDnum (Value [\Size])
```

`Value`

Datový typ: `dnum`

Data dnum v reprezentaci celého čísla.

`Size`

Datový typ: `num`

Velikost (počet bitů), která by měla být brána v úvahu při provádění logické bitové `NEGATION` operace na argumentu `Value`. Velikost je platná od 1 - 52.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.22 BitNegDnum - Logická bitová NEGATION operace na datech dnum

Pokračování

Omezení

Rozsah pro datový typ je `dnum` je 0 - 4503599627370495.

Další příklady

Více příkladů funkce `BitNegDnum` je názorně uvedeno dole.

Příklad 1

```
VAR dnum result;
VAR dnum data1:=38;
! Only consider the 16 lowest bits
result := BitNegDnum(data1 \Size:=16);
TPWrite "" \Dnum:=result;
! Consider all 52 bits in the dnum datatype
result := BitNegDnum(data1);
TPWrite "" \Dnum:=result;
```

Logická bitová `NEGATION` operace bude vykonána na `data1` a výsledek bude vrácen do `result` (reprezentace celého čísla). První hodnota, která bude zapsána na FlexPendant, je 65497. Druhá hodnota, která bude zapsána na FlexPendant, je 4503599627370457.

Syntaxe

```
BitNegDnum '('
  [Value ':='] <expression (IN) of dnum>
  ['\'Size ':=' < expression (IN) of num>]
  ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Logická bitová NEGATION operace na datech byte	*BitNeg - Logická bitová NEGATION - operace na datech byte na str 1061*
Datový typ `dnum`	*dnum - Dvojité numerické hodnoty na str 1485*
Logická bitová AND operace na datech dnum	*BitAndDnum - Logická bitová AND - operace na datech dnum na str 1050*
Logická bitová OR operace na datech dnum	*BitOrDnum - Logická bitová OR operace na datech dnum na str 1067*
Logická bitová XOR operace na datech dnum	*BitXOrDnum - Logická bitová XOR operace na datech dnum na str 1075*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.23 BitOr - Logická bitová OR- operace na datech byte

Použití

`BitOr` (*Bit inclusive Or*) se používá k vykonání logické bitové operace `OR` na datových typech `byte`.

Základní příklady

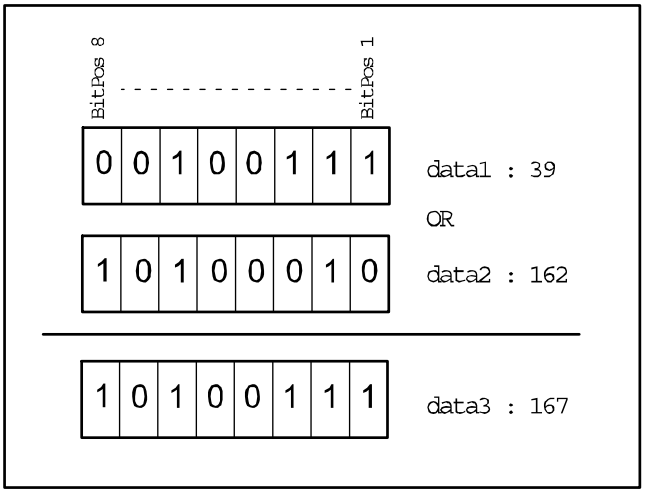
Následující příklad názorně ukazuje funkci`BitOr`.

Příklad 1

```
VAR byte data1 := 39;
VAR byte data2 := 162;
VAR byte data3;

data3 := BitOr(data1, data2);
```

Logická bitová OR operace bude vykonána na `data1` a `data2`. Výsledek bude vrácen do `data3` (reprezentace celého čísla).

Následující obrázek ukazuje logickou bitovou OR operaci.



xx0500002458

Vratná hodnota (Vrátit hodnotu)

Datový typ: `byte`

Výsledek logické bitové OR operace v reprezentaci celého čísla.

Argumenty

```
BitOr (BitData1 BitData2)
```

`BitData1`

Datový typ: `byte`

Bitová data 1 v reprezentaci celého čísla.

`BitData2`

Datový typ: `byte`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.23 BitOr - Logická bitová OR- operace na datech byte

RobotWare - OS

Pokračování

Bitová data 2 v reprezentaci celého čísla.

Omezení

Rozsah pro datový typ je `byte` je 0 - 255.

Syntaxe

```
BitOr '('
  [BitData1 ':='] <expression (IN) of byte> ','
  [BitData2 ':='] <expression (IN) of byte>
  ')'
```

Funkce s vrácenou hodnotou datového typu `byte`.

Související informace

Pro informace o	Viz
Logická bitová AND operace na datech byte	*BitAnd - Logická bitová AND - operace na datech byte na str 1048*
Logická bitová XOR - operace na datech byte	*BitXOr - Logická bitová XOR operace na datech byte na str 1073*
Logická bitová NEGATION - operace na datech byte	*BitNeg - Logická bitová NEGATION - operace na datech byte na str 1061*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.24 BitOrDnum - Logická bitová OR operace na datech dnum

Použití

`BitOrDnum` (*Bit inclusive Or dnum*) se používá k vykonání logické bitové operace `OR` na datových typech `dnum`.

Základní příklady

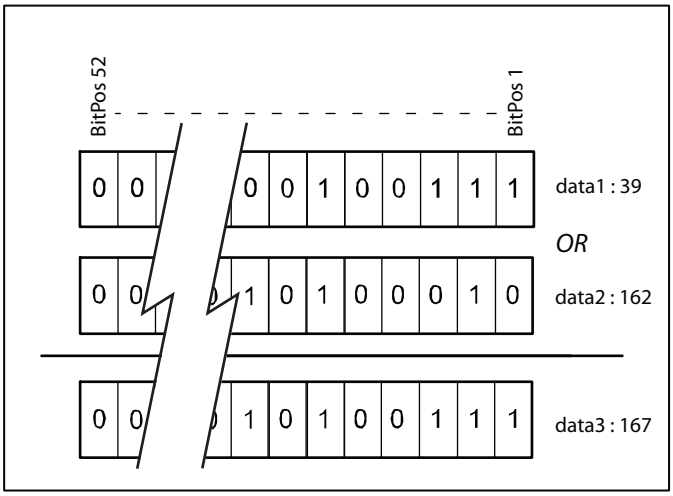
Následující příklad názorně ukazuje funkci`BitOrDnum`.

Příklad 1

```
VAR dnum data1 := 39;
VAR dnum data2 := 162;
VAR dnum data3;

data3 := BitOrDnum(data1, data2);
```

Logická bitová OR operace bude vykonána na `data1` a `data2`. Výsledek bude vrácen do `data3` (reprezentace celého čísla).

Následující obrázek ukazuje logickou bitovou OR operaci.



xx1200000011

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Výsledek logické bitové OR operace v reprezentaci celého čísla.

Argumenty

```
BitOrDnum (Value1 Value2)
```

`Value1`

Datový typ: `dnum`

Datová hodnota prvního bitu v reprezentaci celého čísla.

`Value2`

Datový typ: `dnum`

Datová hodnota druhého bitu v reprezentaci celého čísla.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.24 BitOrDnum - Logická bitová OR operace na datech dnum

Pokračování

Omezení

Rozsah pro datový typ je `dnum` je 0 - 4503599627370495.

Syntaxe

```
BitOrDnum '('
  [Value1 ':='] <expression (IN) of dnum> ','
  [Value2 ':='] <expression (IN) of dnum>
  ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Logická bitová OR - operace na datech **byte**	*BitOr - Logická bitová OR- operace na datech byte na str 1065*
Datový typ `dnum`	*dnum - Dvojité numerické hodnoty na str 1485*
Logická bitová AND - operace na datech `dnum`	*BitAndDnum - Logická bitová AND - operace na datech dnum na str 1050*
Logická bitová XOR - operace na datech `dnum`	*BitXOrDnum - Logická bitová XOR operace na datech dnum na str 1075*
Logická bitová NEGATION - operace na datech `dnum`	*BitNegDnum - Logická bitová NEGATION operace na datech dnum na str 1063*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.25 BitRSh - Logická bitová RIGHT SHIFT operace na bajtech

Použití

`BitRSh` (*Bit Right Shift*) se používá k vykonání logické bitové RIGHT SHIFT operace na datových typech `byte`.

Základní příklady

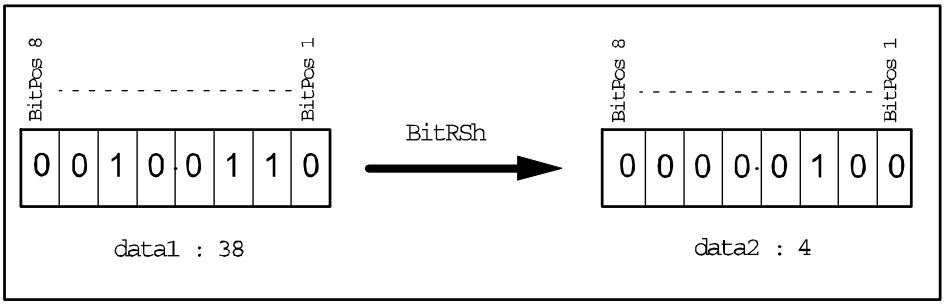
Následující příklad názorně ukazuje funkci`BitRSh`.

Příklad 1

```
VAR num right_shift := 3;
VAR byte data1 := 38;
VAR byte data2;

data2 := BitRSh(data1, right_shift);
```

Logická bitová RIGHT SHIFT operace bude vykonána na `data1` s 3 (`right_shift`) kroky pravého posunu, a výsledek bude vrácen do `data2` (reprezentace celého čísla).

Následující obrázek ukazuje logickou bitovou RIGHT SHIFT operaci.



xx0500002455

Vratná hodnota (Vrátit hodnotu)

Datový typ: `byte`

Výsledek logické bitové RIGHT SHIFT operace v reprezentaci celého čísla.

Levé bitové buňky budou naplněny 0-bity.

Argumenty

```
BitRSh (BitData ShiftSteps)
```

`BitData`

Datový typ: `byte`

Bitová data, v reprezentaci celého čísla, která budou posunuta.

`ShiftSteps`

Datový typ: `num`

Počet logických posunutí (1 - 8), která budou vykonána.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.25 BitRSh - Logická bitová RIGHT SHIFT operace na bajtech

RobotWare - OS

Pokračování

Omezení

Rozsah pro datový typ je `byte` je 0 - 255.

Argument `ShiftSteps` je platný od 1 - 8 podle jednoho bajtu.

Syntaxe

```
BitRSh '('
  [BitData ':='] <expression (IN) of byte> ','
  [ShiftSteps ':='] <expression (IN) of num>
  ')'
```

Funkce s vrácenou hodnotou datového typu byte.

Související informace

Pro informace o	Viz
Logická bitová LEFT SHIFT operace na datech byte	*BitLSh - Logická bitová LEFT SHIFT - operace na bajtech na str 1056*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.26 BitRShDnum - Logická bitová RIGHT SHIFT operace na dnum

Použití

`BitRShDnum` (*Bit Right Shift dnum*) se používá k vykonání logické bitové RIGHT SHIFT operace na datových typech `dnum`.

Základní příklady

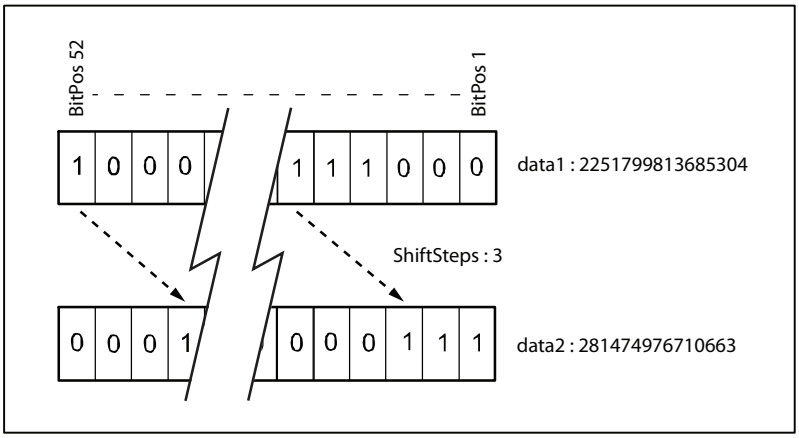
Následující příklad názorně ukazuje funkci`BitRShDnum`.

Příklad 1

```
VAR num right_shift := 3;
VAR dnum data1 := 2251799813685304;
VAR dnum data2;

data2 := BitRShDnum(data1, right_shift);
```

Logická bitová RIGHT SHIFT operace bude vykonána na `data1` s 3 (`right_shift`) kroky pravého posunu, a výsledek bude vrácen do `data2` (reprezentace celého čísla).

Následující obrázek ukazuje logickou bitovou RIGHT SHIFT operaci.



xx1200000009

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Výsledek logické bitové RIGHT SHIFT operace v reprezentaci celého čísla.

Levé bitové buňky budou naplněny 0-bity.

Argumenty

```
BitRShDnum (Value ShiftSteps)
```

`Value`

Datový typ: `dnum`

Bitová data, v reprezentaci celého čísla, která budou posunuta.

`ShiftSteps`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.26 BitRShDnum - Logická bitová RIGHT SHIFT operace na dnum

Pokračování

Počet logických posunutí (1 - 52), která budou vykonána.

Omezení

Rozsah pro datový typ je `dnum` je 0 - 4503599627370495.

Argument `ShiftSteps` je platný od 1 - 52, jelikož jeden dnum je 52 bitů.

Syntaxe

```
BitRShDnum '('
  [Value ':='] <expression (IN) of dnum> ','
  [ShiftSteps ':='] <expression (IN) of num>
  ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Logická bitová RIGHT SHIFT operace na datech **byte**	*BitRSh - Logická bitová RIGHT SHIFT operace na bajtech na str 1069*
Datový typ `dnum`	*dnum - Dvojité numerické hodnoty na str 1485*
Logická bitová LEFT SHIFT operace na datech dnum	*BitLShDnum - Logická bitová LEFT SHIFT operace na dnum na str 1058*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.27 BitXOr - Logická bitová XOR operace na datech byte

Použití

`BitXOr` (*Bit eXclusive Or*) se používá k vykonání logické bitové operace XOR na datových typech `byte`.

Základní příklady

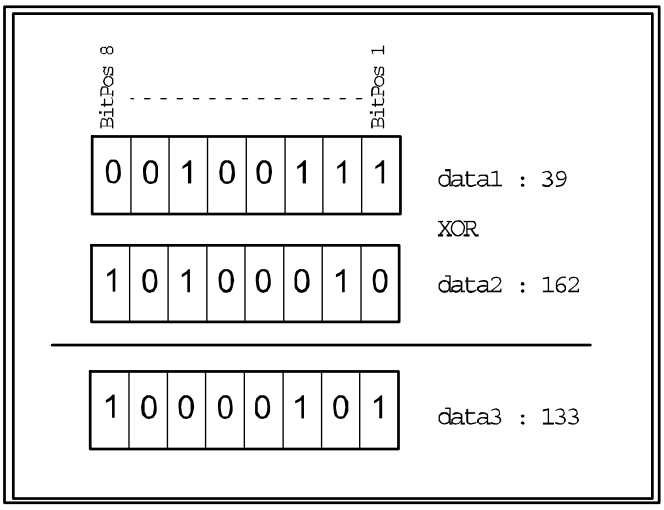
Následující příklad názorně ukazuje funkci`BitXOr`.

Příklad 1

```
VAR byte data1 := 39;
VAR byte data2 := 162;
VAR byte data3;

data3 := BitXOr(data1, data2);
```

Logická bitová XOR operace bude vykonána na `data1` a `data2`. Výsledek bude vrácen do `data3` (reprezentace celého čísla).

Následující obrázek ukazuje logickou bitovou XOR operaci.



xx0500002459

Vratná hodnota (Vrátit hodnotu)

Datový typ: `byte`

Výsledek logické bitové XOR operace v reprezentaci celého čísla.

Argumenty

```
BitXOr (BitData1 BitData2)
```

`BitData1`

Datový typ: `byte`

Bitová data 1 v reprezentaci celého čísla.

`BitData2`

Datový typ: `byte`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.27 BitXOr - Logická bitová XOR operace na datech byte

RobotWare - OS

Pokračování

Bitová data 2 v reprezentaci celého čísla.

Omezení

Rozsah pro datový typ je `byte` je 0 - 255.

Syntaxe

```
BitXOr '('
  [BitData1 ':='] <expression (IN) of byte> ','
  [BitData2 ':='] <expression (IN) of byte>
  ')'
```

Funkce s vrácenou hodnotou datového typu `byte`.

Související informace

Pro informace o	Viz
Logická bitová AND operace na datech byte	*BitAnd - Logická bitová AND - operace na datech byte na str 1048*
Logická bitová AND - operace na datech byte	*BitOr - Logická bitová OR- operace na datech byte na str 1065*
Logická bitová NEGATION - operace na datech byte	*BitNeg - Logická bitová NEGATION - operace na datech byte na str 1061*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.28 BitXOrDnum - Logická bitová XOR operace na datech dnum

Použití

`BitXOrDnum` (*Bit eXclusive Or dnum*) se používá k vykonání logické bitové operace XOR na datových typech `dnum`.

Základní příklady

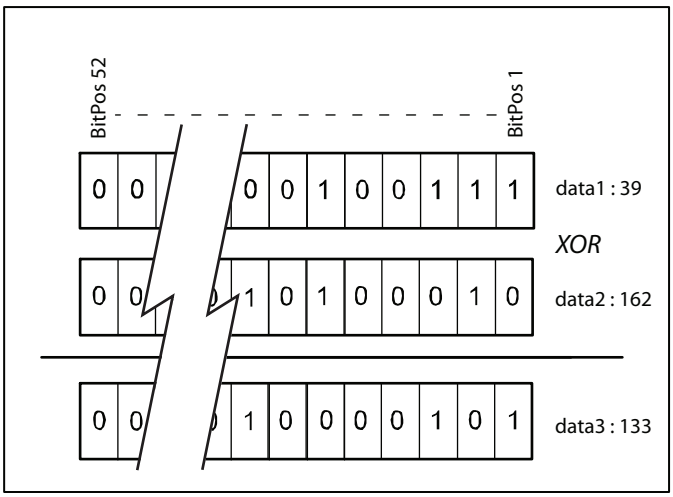
Následující příklad názorně ukazuje funkci`BitXOrDnum`.

Příklad 1

```
VAR dnum data1 := 39;
VAR dnum data2 := 162;
VAR dnum data3;

data3 := BitXOrDnum(data1, data2);
```

Logická bitová XOR operace bude vykonána na `data1` a `data2`. Výsledek bude vrácen do `data3` (reprezentace celého čísla).

Následující obrázek ukazuje logickou bitovou XOR operaci.



xx1200000010

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Výsledek logické bitové XOR operace v reprezentaci celého čísla.

Argumenty

```
BitXOrDnum (Value1 Value2)
```

`Value1`

Datový typ: `dnum`

Datová hodnota prvního bitu v reprezentaci celého čísla.

`Value2`

Datový typ: `dnum`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.28 BitXOrDnum - Logická bitová XOR operace na datech dnum

Pokračování

Datová hodnota druhého bitu v reprezentaci celého čísla.

Omezení

Rozsah pro datový typ je `dnum` je 0 - 4503599627370495.

Syntaxe

```
BitXOrDnum '('
  [Value1 ':='] <expression (IN) of dnum> ','
  [Value2 ':='] <expression (IN) of dnum>
  ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Logická bitová XOR - operace na datech byte	*BitXOr - Logická bitová XOR operace na datech byte na str 1073*
Datový typ `dnum`	*dnum - Dvojité numerické hodnoty na str 1485*
Logická bitová AND operace na datech dnum	*BitAndDnum - Logická bitová AND - operace na datech dnum na str 1050*
Logická bitová OR operace na datech dnum	*BitOrDnum - Logická bitová OR operace na datech dnum na str 1067*
Logická bitová NEGATION operace na datech dnum	*BitNegDnum - Logická bitová NEGATION operace na datech dnum na str 1063*
Další bitové funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.29 ByteToStr - Převádí byte na řetězcová data

Použití

`ByteToStr` (*Byte To String*) se používá k převodu byte na řetězcová data s definovaným datovým formátem byte.

Základní příklady

Následující příklad názorně ukazuje funkci`ByteToStr`.

Příklad 1

```
VAR string con_data_buffer{5};
VAR byte data1 := 122;

con_data_buffer{1} := ByteToStr(data1);
```

Obsah komponentu pole `con_data_buffer{1}` bude "122" po funkci `ByteToStr ....`

```
con_data_buffer{2} := ByteToStr(data1\Hex);
```

Obsah komponentu pole `con_data_buffer{2}` bude "7A" po funkci `ByteToStr ....`

```
con_data_buffer{3} := ByteToStr(data1\Okt);
```

Obsah komponentu pole `con_data_buffer{3}` bude "172" po funkci `ByteToStr ....`

```
con_data_buffer{4} := ByteToStr(data1\Bin);
```

Obsah komponentu pole `con_data_buffer{4}` bude "01111010" po funkci `ByteToStr ....`

```
con_data_buffer{5} := ByteToStr(data1\Char);
```

Obsah komponentu pole `con_data_buffer{5}` bude "z" po funkci `ByteToStr ....`

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Výsledek operace převodu s následujícím formátem:

Formát	Znaky	Délka řetězce	Rozsah
Dec .....:	’0’ -’ 9’	1-3	"0" - "255"
Hex .....:	’0’ -’ 9’, ’A’ -’F’	2	"00" - "FF"
Okt .....:	’0’ - ’7’	3	"000" - "377"
Bin .....:	’0’ - ’1’	8	"00000000" - "11111111"
Char ....:	Všechny znaky ASCII (*)	1	Jeden znak ASCII

(*) Jestliže se jedná o nezapisovatelný znak ASCII, potom bude vratný formát ve znakovém kódu RAPID (například “ \07” pro kontrolní znak BEL).

Argumenty

```
ByteToStr (BitData [\Hex] | [\Okt] | [\Bin] | [\Char])
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.29 ByteToStr - Převádí byte na řetězcová data

RobotWare - OS

Pokračování

`BitData`

Datový typ: `byte`

Bitová data, která budou převedena.

Jestliže je vypuštěn volitelný argument přepínače, data budou převedena do formátu `decimal` (Dec).

`[\Hex]`

Hexadecimal

Datový typ: `switch`

Data budou převedena do formátu `hexadecimal`.

`[\Okt]`

Octal

Datový typ: `switch`

Data budou převedena do formátu `octal`.

`[\Bin]`

Binary

Datový typ: `switch`

Data budou převedena do formátu `binary`.

`[\Char]`

Character

Datový typ: `switch`

Data budou převedena do znakového formátu `ASCII`.

Omezení

Rozsah pro datový typ je `byte` 0 až 255 desítkový.

Syntaxe

```
ByteToStr '('
  [BitData ':='] <expression (IN) of byte>
  ['\' Hex ] | ['\' Okt] | ['\' Bin] | ['\' Char]
  ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Převést řetězec na bajtová data	*StrToByte - Převádí řetězec na bajtová data na str 1351*
Ostatní bitové (bajtové) funkce	*Technická referenční příručka - Přehled RAPID*
Ostatní řetězcové funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.30 CalcJointT - Vypočítává úhly spoje od robtarget

Použití

`CalcJointT` (*Calculate Joint Target*) se používá pro výpočet úhlů spoje os robotu a externích os od určených dat `robtarget`.

Vstupní data `robtarget` by měla být určena ve stejném souřadném systému, jak je určen v argumentu pro `Tool`, `WObj`, a v době vykonávání aktivní posun programu (`ProgDisp`) a ofset externích os (`EOffs`). Vrácená data `jointtarget` jsou vyjádřena v kalibračním souřadném systému.

Jestliže se jedná o polokoordinovaný nebo synchronizovaný koordinovaný režim aplikačního typu MultiMove s koordinovaným pracovním objektem, a byl posunut některou mechanickou jednotkou umístěnou v jiné programové úloze, potom může být použita funkce `CalcJointT`, jestliže:

- Je vhodné, aby aktuální pozice koordinovaného pracovního objektu posunutá mechanickou jednotkou byla použita ve výpočtu (aktuální uživatelský rámec). Všechna ostatní data budou získána z programu RAPID.
- Mechanická jednotka umístěná v jiné programové úloze stojí v klidu.
- Je použit argument `\UseCurWObjPos`.

Základní příklady

Následující příklady názorně ukazují funkci`CalcJointT`.

Příklad 1

```
VAR jointtarget jointpos1;
CONST robtarget p1 := [...];
jointpos1 := CalcJointT(p1, tool1 \WObj:=wobj1);
```

Hodnota `jointtarget` odpovídající `robtarget` hodnotě `p1` je uložena do `jointpos1`. Nástroj `tool1` a pracovní objekt `wobj1` jsou použity pro výpočet úhlů spoje `jointpos1`.

Příklad 2

```
VAR jointtarget jointpos2;
CONST robtarget p2 := [...];
jointpos2 := CalcJointT(\UseCurWObjPos, p2, tool2 \WObj:=orb1);
```

Hodnota `jointtarget` odpovídající `robtarget` hodnotě `p2` je uložena do `jointpos2`. Nástroje `tool2` a pracovní objekt `orb1` jsou použity pro výpočet úhlů spoje `jointpos2`. Aktuální pozice manipulátoru stojícího v klidu `orb1` není umístěna ve stejné programové úloze jako TCP robotu, ale je použita pro výpočet.

Příklad 3

```
VAR jointtarget jointpos3;
CONST robtarget p3 := [...];
VAR errnum myerrnum;
jointpos3 := CalcJointT(p3, tool2 \WObj:=orb1
     \ErrorNumber:=myerrnum);
IF myerrnum = ERR_ROBLIMIT THEN
  TPWrite "Joint jointpos3 can not be reached.";
  TPWrite "jointpos3.robax.rax_1: "+ValToStr(jointpos3.robax.rax_1);
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.30 CalcJointT - Vypočítává úhly spoje od robtarget

RobotWare - OS

Pokračování

```
    ..
    ..
    TPWrite "jointpos3.extax.eax_f"+ValToStr(jointpos3.extax.eax_f);
  ELSEIF myerrnum = ERR_OUTSIDE_REACH THEN
    TPWrite "Joint jointpos3 is outside reach.";
    TPWrite "jointpos3.robax.rax_1: "+ValToStr(jointpos3.robax.rax_1);
    ..
    ..
    TPWrite "jointpos3.extax.eax_f"+ValToStr(jointpos3.extax.eax_f);
  ELSE
    MoveAbsJ jointpos3, v100, fine, tool2 \WObj:=orb1;
  ENDIF
```

Hodnota `jointtarget` **odpovídající** `robtarget` **hodnotě** `p3` **je uložena do** `jointpos3`**. Jestliže je možné této pozice dosáhnout, je použita, jinak je hodnota** `jointtarget` **zapsána na FlexPendant.**

Vratná hodnota (Vrátit hodnotu)

Datový typ: `jointtarget`

Úhly ve stupních pro osy robotu na straně ramena.

Hodnoty pro externí osy, v mm pro lineární osy, ve stupních pro rotační osy.

Vrácené hodnoty vždy souvisí s kalibrační pozicí.

Argumenty

```
CalcJointT ( [\UseCurWObjPos] Rob_target Tool [\WObj]
     [\ErrorNumber])
```

`[\UseCurWObjPos]`

Datový typ: `switch`

Použijte aktuální pozici koordinovaného pracovního objektu posunutého mechanickou jednotkou v jiné úloze pro výpočet (aktuální uživatelský rámec). Všechna ostatní data jsou získána z programu RAPID.

`Rob_target`

Datový typ: `robtarget`

Pozice robotu a externích os v nejvzdálenějším souřadném systému souvisejícím s určeným nástrojem a pracovním objektem a v době vykonávání aktivním posunem programu (`ProgDisp`) **a/nebo ofsetem externích os** (`EOffs`)**.**

`Tool`

Datový typ: `tooldata`

Nástroj použitý k výpočtu úhlů spoje robotu.

`[\WObj]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadný systém), ke kterému se vztahuje pozice robotu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže tento argument je vypuštěn, je použit pracovní objekt `wobj0`. Tento argument musí být určen při používání stacionárního nástroje, koordinovaných externích os nebo dopravníku.

`[\ErrorNumber]`

Error number

Datový typ: `errnum`

Proměnná (`VAR` nebo `PERS`), která bude držet chybovou konstantu `ERR_ROBLIMIT`, jestliže alespoň jedna osa je mimo limity spoje nebo jestliže limity jsou překročeny alespoň u jednoho spojeného spoje, nebo `ERR_OUTSIDE_REACH`, jestliže pozice (robtarget) je mimo pracovní oblast robotu. Jestliže se použije tento volitelný argument a proměnná je nastavena na `ERR_ROBLIMIT` nebo `ERR_OUTSIDE_REACH` po vykonání funkce, vrácená hodnota bude `jointtarget` hodnota odpovídající použitému `robtarget`.

Jestliže tato volitelná proměnná je vynechána, bude vykonán chybový handler a vrácený `jointarget` nebude aktualizován, jestliže osa je mimo pracovní oblast nebo limity jsou překročeny.

Vykonávání programu

Vrácený `jointtarget` je vypočítán ze vstupu `robtarget`. Jestliže je použit argument `\UseCurWObjPos`, potom pozice, která se používá, vychází z aktuální pozice mechanické jednotky, která kontroluje uživatelský rámec. Aby bylo možné vypočítat úhly spoje robotu, musí být vzat v úvahu určený `Tool`, `WObj` (včetně koordinovaného uživatelského rámce, a `ProgDisp`, který je aktivní v době vykonávání. Aby bylo možné vypočítat pozici externích os v době vykonávání, je vzat v úvahu aktivní `EOffs`.

Kalkulace vždy vybírá konfiguraci robotu podle určených konfiguračních dat ve vstupních datech `robtarget`. Instrukce `ConfL` a `ConfJ` neovlivňují tuto zásadu kalkulace. Jestliže je použita singularita zápěstí, osa 4 robotu bude nastavena na 0 stupňů.

Jestliže je zde jakýkoliv posun programu (`ProgDisp`) a/nebo ofset externí osy (`EOffs`) v době, kdy robtarget je uložen, potom stejný posun programu a/nebo ofset externí osy musí být aktivní, když je vykonáván `CalcJointT`.

Omezení

Jestliže je použit rámec souřadnice, musí být aktivována koordinovaná jednotka před použitím `CalcJointT`.

Mechanická jednotka, která kontroluje uživatelský rámec v pracovním objektu, musí být normálně dostupná ve stejné programové úloze jako TCP robotu, který vykonává `CalcJointT`.

Normálně `CalcJointT` používá `robtarget`, `tooldata`, a `wobjdata` z programu RAPID pro výpočet `jointtarget`. U koordinovaných pracovních objektů je pozice mechanické jednotky dána pozice externích os v `robtarget`. To ale není ten případ, jestliže mechanická jednotka je kontrolována jinou programovou úlohou (systém MultiMove) nebo mechanická jednotka není kontrolována kontrolním systémem (Dopravník). U systému MultiMove, ale nikoliv u dopravníku je možné

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.30 CalcJointT - Vypočítává úhly spoje od robtarget

RobotWare - OS

Pokračování

použít argument `\UseCurWObjPos`, jestliže mechanická jednotka stojí v klidu v době vykonávání `CalCJointT`.

Řešení chyb

Jestliže pozice je dosažitelná, ale nejméně jedna osa je mimo limity spoje nebo limity jsou překročeny u nejméně jednoho spojeného spoje, potom systémová proměnná `ERRNO` je nastavena na `ERR_ROBLIMIT` a vykonávání pokračuje v chybovém handleru.

Jestliže pozice (robtarget) je mimo pracovní rozsah robotu, potom systémová proměnná `ERRNO` je nastavena na `ERR_OUTSIDE_REACH` a vykonávání pokračuje v chybovém handleru.

Jestliže mechanická jednotka, která kontroluje pracovní objekt (uživatelský rámec), nestojí v klidu v době vykonávání `CalJointT \UseCurWobjPos`, potom systémová proměnná `ERRNO` je nastavena na `ERR_WOBJ_MOVING` a vykonávání pokračuje v chybovém handleru.

Chybový handler se může potom vypořádat s těmito situacemi.

Syntaxe

```
CalcJointT '('
  [ '\'UseCurWObjPos ',']
  [ Rob_target ':='] <expression (IN) of robtarget> ','
  [ Tool ':=' ] <persistent (PERS) of tooldata>
  [ '\' WObj ':=' <persistent (PERS) of wobjdata>]
  [ '\' ErrorNumber ':=' <variable or persistent (INOUT) of errnum>]
  ')'
```

Funkce s vrácenou hodnotou datového typu jointtarget.

Související informace

Pro informace o	Viz
Vypočítat robtarget z jointtarget	*CalcRobT - Vypočítává robtarget z jointtarget na str 1083*
Definice pozice	*robtarget - Poziční data na str 1572*
Definice pozice spoje	*jointtarget - Data pozice svaru na str 1520*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Souřadný systém posunutí programu	*PDispOn - Aktivuje posun programu na str 477*
Souřadný systém ofsetu externí osy	*EOffsOn - Aktivuje ofset pro pomocné osy na str 197*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.31 CalcRobT - Vypočítává robtarget z jointtarget

Použití

`CalcRobT` (*Calculate Robot Target*) se používá pro výpočet dat `robtarget` z daných dat `jointtarget`.

Tato funkce vrací hodnotu `robtarget` s pozicí (x, y, z), orientací (q1 ... q4), konfigurací os robotu a pozice externích os.

Vstupní data `jointtarget` by měla být určena v souřadném systému kalibrace.

Vrácená data `robtarget` jsou vyjádřena v nejvzdálenějším souřadném systému. Bere se v úvahu určený nástroj, pracovní objekt a v době vykonávání aktivní posun programu (`ProgDisp`) a ofset externí osy (`EOffs`).

Základní příklady

Následující příklad názorně ukazuje funkci`CalcRobT`.

Příklad 1

```
VAR robtarget p1;
CONST jointtarget jointpos1 := [...];

p1 := CalcRobT(jointpos1, tool1 \WObj:=wobj1);
```

Hodnota `robtarget`odpovídající `jointtarget`hodnotě `jointpos1`je uložena do `p1`. Nástroj `tool1` a pracovní objekt `wobj1` jsou použity pro výpočet pozice `p1`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `robtarget`

Pozice robotu a externích os je vrácena v datovém typu `robtarget` a vyjádřena v nejvzdálenějším souřadném systému. Bere se v úvahu určený nástroj, pracovní objekt a v době vykonávání aktivní posun programu (`ProgDisp`) a ofset externí osy (`EOffs`).

Jestliže zde není aktivní `ProgDisp`, potom je pozice robotu vyjádřena v souřadném systému objektu. Jestliže zde nejsou žádné aktivní `EOffs`, potom je pozice externí osy vyjádřena v souřadném systému kalibrace.

Argumenty

```
CalcRobT( Joint_target Tool [\WObj] )
```

`Joint_target`

Datový typ: `jointtarget`

Pozice spoje pro osy robotu a externí osy ve vztahu k souřadnému systému kalibrace.

`Tool`

Datový typ: `tooldata`

Nástroj použitý pro výpočet pozice robotu.

`[\WObj]`

Work Object

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.31 CalcRobT - Vypočítává robtarget z jointtarget

RobotWare - OS

Pokračování

Datový typ: `wobjdata`

Pracovní objekt (souřadný systém), ke kterému se vztahuje pozice robotu vrácená funkcí.

Jestliže tento argument je vypuštěn, je použit pracovní objekt `wobj0`. Tento argument musí být určen při používání stacionárního nástroje, koordinovaných externích os nebo dopravníku.

Vykonávání programu

Vrácený `robtarget` je vypočítán ze vstupu `jointtarget`. Pro výpočet karteziánské pozice robotu je vzat v úvahu určený `Tool`, `WObj` (včetně koordinovaného uživatelského rámce) a v době vykonávání aktivního `ProgDisp`.

Pro výpočet pozice externí osy je také vzat v úvahu `EOffs`, aktivní v době vykonávání.

Omezení

Jestliže je použit koordinovaný rámec, potom musí být aktivována koordinovaná jednotka před použitím `CalcRobT`. Koordinovaná jednotka musí být také situována ve stejné úloze jako robot.

Syntaxe

```
CalcRobT '('
  [Joint_target ':=' ] <expression (IN) of jointtarget> ','
  [Tool ':=' ] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>] ')'
```

Funkce s vrácenou hodnotou datového typu `robtarget`.

Související informace

Pro informace o	Viz
Vypočítat jointtarget z robtarget	*CalcJointT - Vypočítává úhly spoje od robtarget na str 1079*
Definice pozice	*robtarget - Poziční data na str 1572*
Definice pozice spoje	*jointtarget - Data pozice svaru na str 1520*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Souřadný systém posunutí programu	*PDispOn - Aktivuje posun programu na str 477*
Souřadný systém ofsetu externích os	*EOffsOn - Aktivuje ofset pro pomocné osy na str 197*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

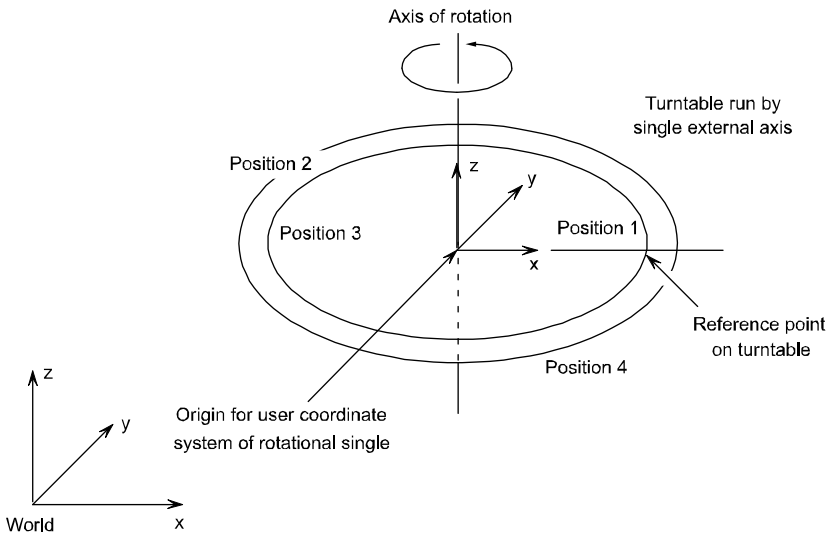
2.32 CalcRotAxFrameZ - Vypočítat rámec rotační osy

Použití

`CalcRotAxFrameZ` (*Calculate Rotational Axis Frame with positive Z-point*) se používá k výpočtu uživatelského souřadného systému mechanické jednotky, která je typu rotační osy. Tato funkce se použije, když řídicí robot a pomocná osa jsou umístěny v různých úlohách RAPID. Jestliže jsou ve stejné úloze, potom by se měla použít funkce `CalcRotAxisFrame`.

Popis

Definice uživatelského rámce pro rotační externí osu vyžaduje, aby otočný stůl (nebo podobná mechanická konstrukce) na externí ose měl označený referenční bod. Navíc, TCP rámu základny robotu a TCP musí být kalibrovány. Kalibrační procedura se skládá v řady pozic pro TCP robotu na referenčním bodu, když je otočný stůl otáčen do různých úhlů. Polohování TCP robotu v kladném směru z je rovněž nutné. Definici bodů pro rotační osu najdete na obrázku dole.



xx0500002468

Uživatelský souřadný systém pro rotační osu má svůj počátek ve středu otočného stolu. Směr z se shoduje s osou rotace a osa x prochází referenčním bodem.

Pokračování na další straně

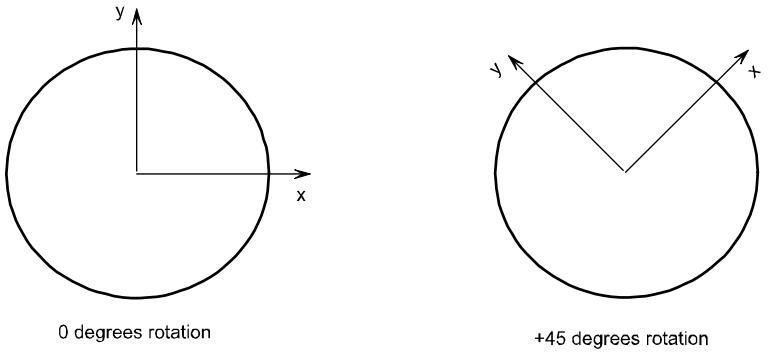
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.32 CalcRotAxFrameZ - Vypočítat rámec rotační osy

RobotWare - OS

Pokračování

Obrázek dole ukazuje uživatelský souřadný systém pro dvě odlišné pozice otočného stolu (otočný stůl je zobrazen shora).



xx0500002469

Základní příklady

Následující příklad názorně ukazuje funkci`CalcRotAxFrameZ`.

Příklad 1

```
CONST robtarget pos1 := [...];
CONST robtarget pos2 := [...];
CONST robtarget pos3 := [...];
CONST robtarget pos4 := [...];
CONST robtarget zpos;
VAR robtarget targetlist{10};
VAR num max_err := 0;
VAR num mean_err := 0;
VAR pose resFr:=[...];
PERS tooldata tMyTool:= [...];

! Instructions for creating/ModPos pos1 - pos4 with TCP pointing
     at the turntable.
MoveJ pos1, v10, fine, tMyTool;
MoveJ pos2, v10, fine, tMyTool;
MoveJ pos3, v10, fine, tMyTool;
MoveJ pos4, v10, fine, tMyTool;

!Instruction for creating/ModPos zpos with TCP pointing at a point
     in positive z direction
MoveJ zpos, v10, fine, tMyTool;

! Add the targets to the array
targetlist{1}:= pos1;
targetlist{2}:= pos2;
targetlist{3}:= pos3;
targetlist{4}:= pos4;

resFr:=CalcRotAxFrameZ(targetlist, 4, zpos, max_err, mean_err);

! Update the system parameters.
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

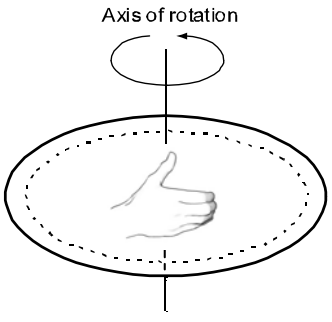
```
IF (max_err < 1.0) AND (mean_err < 0.5) THEN
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_x",
       resFr.trans.x/1000;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_y",
       resFr.trans.y/1000;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_z",
       resFr.trans.z/1000;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u0",
       resFr.rot.q1;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u1",
       resFr.rot.q2;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u2",
       resFr.rot.q3;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u3",
       resFr.rot.q4;
  TPReadFK reg1,"Warmstart required for calibration to take
       effect.", stEmpty, stEmpty, stEmpty, stEmpty,"OK";
  WarmStart;
ENDIF
```

Čtyři pozice `pos1` - `pos4` jsou vytvořeny/modposed tak, že nástroj robotu `tMyTool` ukazuje ke stejném referenčnímu bodu na externí ose `STN_1`, ale s různými rotacemi externí osy. Pozice `zpos` je vytvořena/modposed tak, že nástroj robotu `tMyTool` ukazuje do kladného směru z podle definice kladného směru z externí rotační mechanické jednotky. Použití definice pozitivního směru z externí rotační mechanické jednotky, viz *Popis na str 1085*. Body jsou potom použity pro výpočet základnového rámu externí osy `resFr`, ve vztahu ke světovému souřadnému systému. Konečně, rám je zapsán do konfiguračního souboru a instrukce `WarmStart` je vykonána, aby změna byla účinná.

POZNÁMKA

Definice kladného směru z externí rotační mechanické jednotky:

Nechte prsty pravé ruky krýt se s kladnou rotační osou rotační osy. Směr palce potom definuje kladný směr z. Viz následující obrázek.



xx0500002472

Vratná hodnota (Vrátit hodnotu)

Datový typ: `pose`

Vypočítaný rámec.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.32 CalcRotAxFrameZ - Vypočítat rámec rotační osy

RobotWare - OS

Pokračování

Argumenty

```
CalcRotAxFrameZ (TargetList TargetsInList PositiveZPoint
    MaxErrMeanErr)
```

`TargetList`

Datový typ: `robtarget`

Pole robtarget držící pozice definované ukazováním ven od otočného stolu. Min počet robtarget je 4, maximum je 10.

`TargetsInList`

Datový typ: `num`

Počet robtarget v poli.

`PositiveZPoint`

Datový typ: `robtarget`

`robtarget` **držící pozici definovanou ukazováním ven a ukazováním dovnitř kladného směru z. Používání definice kladného směru z externí rotační mechanické jednotky - viz** *Popis na str 1085*.

`MaxErr`

Maximum Error

Datový typ: `num`

Odhadovaná maximální chyba v mm.

`MeanErr`

Mean Error

Datový typ: `num`

Odhadovaná střední chyba v mm.

Řešení chyb

Jestliže pozice nemají požadovaný vztah nebo nejsou určeny s dostatečnou přesností, potom je systémová proměnná `ERRNO` **je nastavena na** `ERR_FRAME` **. Tuto chybu je možné potom ošetřit v chybovém handleru.**

Syntaxe

```
CalcRotAxFrameZ '('
  [TargetList ':='] <array {*} (IN) of robtarget> ','
  [TargetsInList ':='] <expression (IN) of num> ','
  [PositiveZPoint ':='] <expression (IN) of robtarget> ','
  [MaxErr ':='] <variable (VAR) of num> ','
  [MeanErr ':='] <variable (VAR) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `pose`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

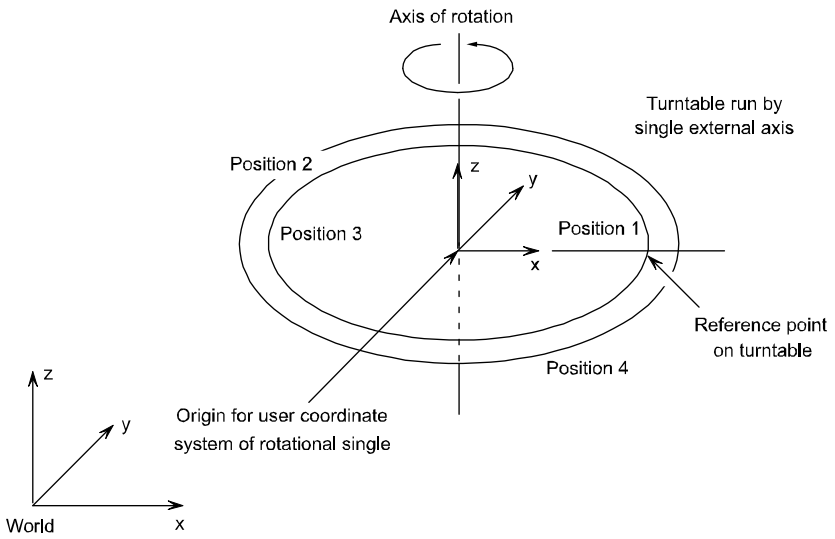
2.33 CalcRotAxisFrame - Vypočítat rámec rotační osy

Použití

`CalcRotAxisFrame` (*Calculate Rotational Axis Frame*) se používá k výpočtu uživatelského souřadného systému mechanické jednotky, která je typu rotační osy. Tato funkce se použije, když řídicí robot a pomocná osa jsou umístěny ve stejné úloze RAPID. Jestliže jsou v různých úlohách, potom by se měla použít funkce `CalcRotAxFrameZ`.

Popis

Definice uživatelského rámce pro rotační externí osu vyžaduje, aby otočný stůl (nebo podobná mechanická konstrukce) na externí ose měl označený referenční bod. Navíc, základnový rám řídicího robotu a TCP musí být kalibrovány. Kalibrační procedura se skládá v řady pozic pro TCP robotu na referenčním bodu, když je otočný stůl otáčen do různých úhlů. Definici bodů pro rotační osu najdete na obrázku dole.



xx0500002468

Uživatelský souřadný systém pro rotační osu má svůj počátek ve středu otočného stolu. Směr z se shoduje s osou rotace a osa x prochází referenčním bodem.

Pokračování na další straně

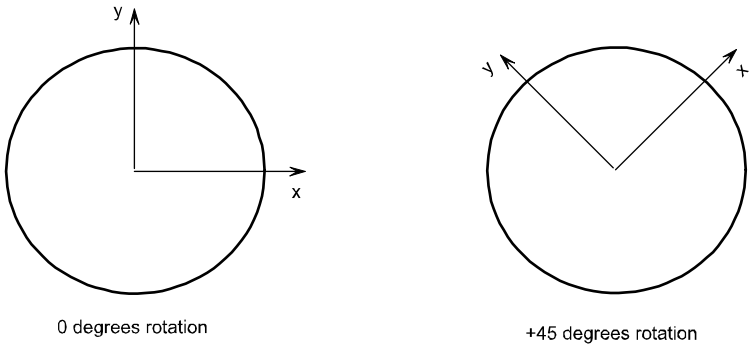
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.33 CalcRotAxisFrame - Vypočítat rámec rotační osy

RobotWare - OS

Pokračování

Obrázek dole ukazuje uživatelský souřadný systém pro dvě odlišné pozice otočného stolu (otočný stůl je zobrazen shora).



xx0500002469

Základní příklady

Následující příklad názorně ukazuje funkci`CalcRotAxisFrame`.

Příklad 1

```
CONST robtarget pos1 := [...];
CONST robtarget pos2 := [...];
CONST robtarget pos3 := [...];
CONST robtarget pos4 := [...];
VAR robtarget targetlist{10};
VAR num max_err := 0;
VAR num mean_err := 0;
VAR pose resFr:=[...];
PERS tooldata tMyTool:= [...];

! Instructions needed for creating/ModPos pos1 - pos4 with TCP
     pointing at the turntable.
MoveJ pos1, v10, fine, tMyTool;
MoveJ pos2, v10, fine, tMyTool;
MoveJ pos3, v10, fine, tMyTool;
MoveJ pos4, v10, fine, tMyTool;

! Add the targets to the array
targetlist{1}:= pos1;
targetlist{2}:= pos2;
targetlist{3}:= pos3;
targetlist{4}:= pos4;

resFr:=CalcRotAxisFrame(STN_1 , targetlist, 4, max_err, mean_err);

! Update the system parameters.
IF (max_err < 1.0) AND (mean_err < 0.5) THEN
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_x",
       resFr.trans.x/1000;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_y",
       resFr.trans.y/1000;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
    WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_z",
         resFr.trans.z/1000;
    WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u0",
         resFr.rot.q1;
    WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u1",
         resFr.rot.q2;
    WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u2",
         resFr.rot.q3;
    WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u3",
         resFr.rot.q4;
    TPReadFK reg1,"Warmstart required for calibration to take
         effect.", stEmpty, stEmpty, stEmpty, stEmpty, "OK";
    WarmStart;
  ENDIF
```

Čtyři pozice, `pos1` - `pos4`, jsou vytvořeny/modposed tak, že nástroj robotu `tMyTool` ukazuje ke stejnému referenčnímu bodu na externí ose `STN_1`, ale s odlišnými rotacemi externí osy. Body jsou potom použity pro výpočet základnového rámu externí osy, `resFr`, ve vztahu ke světovému souřadnému systému. Nakonec je rám zapsán do konfiguračního souboru a instrukce `WarmStart` je vykonána, aby změna byla účinná.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `pose`

Vypočítaný rámec.

Argumenty

```
CalcRotAxisFrame (MechUnit [\AxisNo] TargetList TargetsInList MaxErr
     MeanErr)
```

`MechUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky, která má být kalibrována.

`[\AxisNo]`

Datový typ: `num`

Volitelný argument definující číslo osy, pro kterou by měl být určen rámec. Výchozí hodnota je 1 a vztahuje se k jednoduché rotační ose. U mechanických jednotek s několika osami by číslo osy mělo být dodáno s tímto argumentem.

`TargetList`

Datový typ: `robtarget`

Pole robtarget držící pozice definované ukazováním ven od otočného stolu. Min počet robtarget je 4, maximum je 10.

`TargetsInList`

Datový typ: `num`

Počet robtarget v poli.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.33 CalcRotAxisFrame - Vypočítat rámec rotační osy

RobotWare - OS

Pokračování

`MaxErr`

Maximum Error

Datový typ: `num`

Odhadovaná maximální chyba v mm.

`MeanErr`

Mean Error

Datový typ: `num`

Odhadovaná střední chyba v mm.

Řešení chyb

Jestliže pozice nemají požadovaný vztah nebo nejsou určeny s dostatečnou přesností, potom je systémová proměnná `ERRNO` je nastavena na `ERR_FRAME` . Tuto chybu je možné potom ošetřit v chybovém handleru.

Syntaxe

```
CalcRotAxisFrame '('
  [MechUnit ':='] <variable (VAR) of mecunit>
  [\AxisNo ':=' <expression (IN) of num> ] ','
  [TargetList ':='] <array {*} (IN) of robtarget> ','
  [TargetsInList ':='] <expression (IN) of num> ','
  [MaxErr ':='] <variable (VAR) of num> ','
  [MeanErr ':='] <variable (VAR) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `pose`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.34 CamGetExposure - Získat data podle kamery

Použití

`CamGetExposure` (`Camera Get Exposure`) je funkce, která čte aktuální nastavení pro kameru. S touto funkcí a s instrukcí `CamSetExposure` je možné adaptovat obrázky kamery podle prostředí při běhu.

Základní příklady

Následující příklad názorně ukazuje funkci`CamGetExposure`.

Příklad 1

```
VAR num exposuretime;
...
exposuretime:=CamGetExposure(mycamera \ExposureTime);
IF exposuretime = 10 THEN
  CamSetExposure mycamera \ExposureTime:=9.5;
ENDIF
```

Přikázat kameře `mycamera` změnit čas expozice na 9,5 ms, jestliže aktuální nastavení je 10 ms.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Jedno z nastavení čas expozice, jas nebo kontrast vrácené od kamery jako numerická hodnota.

Argumenty

```
CamGetExposure (Camera [\ExposureTime] | [\Brightness] |
     [\Contrast])
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

`[\ExposureTime]`

Datový typ: `num`

Vrací čas expozice kamery. Hodnota je v milisekundách (ms).

`[\Brightness]`

Datový typ: `num`

Vrací nastavení jasu kamery

`[\Contrast]`

Datový typ: `num`

Vrací nastavení kontrastu kamery

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.34 CamGetExposure - Získat data podle kamery

Integrated Vision
Pokračování

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_CAM_BUSY	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.
ERR_CAM_COM_TIMEOUT	Chyba komunikace s kamerou. Kamera je pravděpodobně odpojena.

Syntaxe

```
CamGetExposure '('
  [ Camera ':=' ] < variable (VAR) of cameradev >
  ['\'ExposureTime]
  | ['\'Brightness]
  | ['\'Contrast] ')'
```

Funkce s vrácenou hodnotou datového typu num.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.35 CamGetLoadedJob - Získat jméno načtené úlohy kamery

Použití

`CamGetLoadedJob` *(Camera Get Loaded Job)* je funkce, která čte jméno aktuální načtené práce od kamery a vrací ji v řetězci.

Základní příklady

Následující příklad názorně ukazuje funkci`CamGetLoadedJob`.

Příklad 1

```
VAR string currentjob;
...
currentjob:=CamGetLoadedJob(mycamera);
IF CurrentJob = "" THEN
  TPWrite "No job loaded in camera "+CamGetName(mycamera);
ELSE
  TPWrite "Job "+CurrentJob+" is loaded in camera "
       "+CamGetName(mycamera);
ENDIF
```

Zapsal jméno načtené práce na FlexPendant.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Aktuální načtené jméno práce pro určenou kameru.

Argumenty

```
CamGetLoadedJob (Camera)
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

Vykonávání programu

Funkce `CamGetLoadedJob` získává aktuálně načtené jméno práce pro kameru. Jestliže není do kamery načtena žádná práce, vrátí se prázdný řetězec.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_CAM_BUSY`	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.
`ERR_CAM_COM_TIMEOUT`	Chyba komunikace s kamerou. Kamera je pravděpodobně odpojena.

Syntaxe

```
CamGetLoadedJob '('
  [ Camera ':=' ] < variable (VAR) of cameradev > ')'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.35 CamGetLoadedJob - Získat jméno načtené úlohy kamery

Integrated Vision

Pokračování

Funkce s vrácenou hodnotou datového typu `string`.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.36 CamGetName - Získat jméno použité kamery

Použití

`CamGetName` (`Camera Get Name`) se používá pro získání konfigurovaného jména kamery.

Základní příklady

Následující příklad názorně ukazuje funkci`CamGetName`.

Příklad 1

```
  ...
  logcameraname camera1;
  CamReqImage camera1;
  ...
  logcameraname camera2;
  CamReqImage camera2;
  ...
PROC logcameraname(VAR cameradev camdev)
  TPWrite "Now using camera: "+CamGetName(camdev);
ENDPROC
```

Procedura zapisuje jméno aktuálně použité kamery do FlexPendant.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Jméno aktuálně použité kamery vrácené jako řetězec.

Argumenty

```
CamGetName(Camera)
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

Syntaxe

```
CamGetName( '('
  [ Camera ':=' ] < variable (VAR) of cameradev > ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.37 CamNumberOfResults - Získat dostupné výsledky

Použití

`CamNumberOfResults (Camera Number of Results)` je funkce, která čte dostupné vizuální výsledky a vrací je jako numerickou hodnotu.

Základní příklady

Následující příklad názorně ukazuje funkci`CamNumberOfResults`.

Příklad 1

```
VAR num foundparts;
...
CamReqImage mycamera;
WaitTime 1;
FoundParts := CamNumberOfResults(mycamera);
TPWrite "Number of identified parts in the camera image:
     "\Num:=foundparts;
```

Zachyťte obrázek. Čekejte na dokončení zpracování obrázku, v tomto případě 1 sekundu. Přečtěte počet identifikovaných kusů a zapište ho do FlexPendant.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Vrací výsledky do souboru pro určenou kameru.

Argumenty

```
CamNumberOfResults (Camera [\SceneId])
```

`Camera`

Datový typ: `cameradev`

Jméno kamery

`[\SceneId]`

Identifikace scény

Datový typ: `num`

`SceneId` je identifikátor, který stanoví, od kterého obrázku číst počet identifikovaných kusů.

Vykonávání programu

`CamNumberOfResults` je funkce, která čte počet dostupných vizuálních výsledků a vrací je jako numerickou hodnotu. Může se používat jako smyčka všemi dostupnými výsledky.

Funkce vrací úroveň fronty přímo, když je funkce vykonávána. Jestliže je funkce vykonána přímo po vyžádání obrázku, výsledek je často 0, protože kamera dosud nedokončila zpracování obrázku.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná ERRNO bude nastavena na:

Název	Příčina chyby
ERR_CAM_BUSY	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.

Syntaxe

```
CamNumberOfResults '('
  [ Camera ':=' ] < variable (VAR) of cameradev >
  [ '\'SceneId ':=' < expression (IN) of num > ] ')'
```

Funkce s vrácenou hodnotou datového typu num.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.38 CapGetFailSigs - Získat vadné I/O signály

Použití

`CapGetFailSigs` se používá pro vracení jmen signálu nebo signálů, které selhaly během dohlížení `CapL` nebo `CapC`.

Jestliže dohled jednoho nebo několika signálů selže během procesu, odstranitelná chyba bude vrácena z instrukce `CapL`/`CapC`. Aby bylo možné určit, který signál nebo signály selhaly, může se použít `CapGetFailSigs` v chybovém handleru pro všechny případy chyb dohledu.

Základní příklad

```
Stringcopied := CapGetFailSigs(Failstring);
```

Pro `Stringcopied` je přidělena hodnota TRUE, jestliže kopie je úspěšná, a FALSE, jestliže selže.

`Failstring` obsahuje signály, které selhaly jako text. Jestliže není možné kopírovat žádný řetězec, je vrácen řetězec `EMPTY`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

TRUE nebo FALSE podle toho, jestli vadný řetězec je modifikován.

Argumenty

```
CapGetFailSigs (ErrorNames)
```

ErrorNames

Datový typ: `string`

`CapGetFailSigs` vyžaduje řetězcovou proměnnou jako vstupní parametr.

Omezení

Jestliže ze seznamu dohledu selhalo větší množství signálů ve stejném čase, pouze tři z nich jsou hlášeny s `CapGetFailSigs`.

Syntaxe

```
CapGetFailSigs '('
[ErrorNames ':='] < variable (INOUT) of string >')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instrukce `InitSuperv`	*InitSuperv - Resetovat veškerý dohled pro CAP na str 271*
Instrukce `SetupSuperv`	*SetupSuperv - Nastavit podmínky pro dohled signálu v CAP na str 639*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Instrukce `RemoveSuperv`	*RemoveSuperv - Odstranit podmínku pro jeden signál na str 538*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.39 CDate - Načíst aktuální datum jako řetězec

Použití

`CDate` (*Current Date*) se používá pro čtení aktuálního datumu systému.

Tuto funkci je možné používat k předložení aktuálního datumu operátorovi na displeji FlexPendantu nebo k poslání aktuálního datumu do textového souboru, do kterého program zapisuje.

Základní příklady

Následující příklad názorně ukazuje funkci`CDate`.

Viz také *Další příklady na str 1102*.

Příklad 1

```
VAR string date;
date := CDate();
```

Aktuální datum je uloženo do proměnné `date`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Aktuální datum v řetězci.

Standardní datový formát je “rok-měsíc-den”, například ”1998-01-29”.

Další příklady

Více příkladů funkce `CDate` je názorně uvedeno dole.

Příklad 1

```
VAR string date;
date := CDate();
TPWrite "The current date is: "+date;
Write logfile, date;
```

Aktuální datum je zapsáno na displej FlexPendantu a do textového souboru.

Syntaxe

```
CDate '(' ')'
```

Funkce s vrácenou hodnotou typu `string`.

Související informace

Pro informace o	Viz
Časové funkce	*Technická referenční příručka - Přehled RAPID*
Nastavování systémových hodin	*Návod k použití - IRC5 s jednotkou FlexPendant*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.40 CJointT - Čte aktuální úhly spoje

Použití

`CJointT` (*Current Joint Target*) se používá pro čtení aktuálních úhlů os robotu a externích os.

Základní příklady

Následující příklad názorně ukazuje funkci`CJointT`.

Viz také *Další příklady na str 1103*.

Příklad 1

```
VAR jointtarget joints;
joints := CJointT();
```

Aktuální úhly os pro robot a externí osy jsou uloženy v `joints`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `jointtarget`

Aktuální úhly ve stupních pro osy robotu na straně ramena.

Aktuální hodnoty pro externí osy, v mm pro lineární osy, ve stupních pro rotační osy.

Vrácené hodnoty souvisí s kalibrační pozicí.

Argumenty

```
CJointT ([\TaskRef]|[\TaskName])
```

`[\TaskRef]`

Task Reference

Datový typ: `taskid`

Identita programové úlohy, ze které by měl být přečten `jointtarget`.

Pro všechny programové úlohy v systému budou dostupné předdefinované proměnné datového typu `taskid`. Identita proměnné bude "taskname"+"Id", například, pro úlohu `T_ROB1`, a identita proměnné bude `T_ROB1Id`.

`[\TaskName]`

Datový typ: `string`

Jméno programové úlohy, ze které by měl být přečten `jointtarget`.

Jestliže žádný z argumentů `\TaskRef` or `\TaskName` není určen, potom se použije aktuální úloha.

Další příklady

Více příkladů funkce `CJointT` je názorně uvedeno dole.

Příklad 1

```
! In task T_ROB1
VAR jointtarget joints;
joints := CJointT(\TaskRef:=T_ROB2Id);
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.40 CJointT - Čte aktuální úhly spoje

RobotWare - OS

Pokračování

Aktuální pozice robotu a externích os v úloze `T_ROB2` jsou uloženy do `joints` v úloze `T_ROB1`.

Všimněte si, že robot v úloze `T_ROB2` se může pohybovat, když je pozice čtena. Aby bylo zajištěno, že robot bude stát v klidu, může být naprogramován stop bod `fine` v předchozí pohybové instrukci v úloze `T_ROB2` a instrukce `WaitSyncTask` může být použita k synchronizaci instrukcí v úloze `T_ROB1`.

Příklad 2

```
! In task T_ROB1
VAR jointtarget joints;
joints := CJointT(\TaskName:="T_ROB2");
```

Stejný účinek jako Příklad 1 nahoře.

Řešení chyb

Jestliže argument `\TaskRef` nebo `\TaskName` určuje některou nepohybovou úlohu, potom je systémová proměnná `ERRNO` nastavena na `ERR_NOT_MOVETASK`. Tato chyba může být ošetřena v chybovém handleru.

Ale žádná chyba nebude generována, jestliže argument `\TaskRef` nebo `\TaskName` určují nepohybovou úlohu, která vykonává tuto funkci `CJointT` (reference k mé vlastní nepohybové úloze). Pozice bude potom získána od připojené pohybové úlohy.

Syntaxe

```
CJointT '('
  ['\' TaskRef ':=' <variable (VAR) of taskid>]
  |['\' TaskName ':=' <expression (IN) of string>] ')'
```

Funkce s vrácenou hodnotou datového typu `jointtarget`.

Související informace

Pro informace o	Viz
Definice spoje	*jointtarget - Data pozice svaru na str 1520*
Čtení aktuálního úhlu motoru	*ReadMotor - Čte aktuální úhly motoru na str 1286*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.41 ClkRead - Čte hodiny použité pro časování

Použití

`ClkRead` se používá ke čtení hodin, které fungují jako stopky používané pro časování.

Základní příklady

Následující příklady názorně ukazují funkci`ClkRead`.

Příklad 1

```
reg1:=ClkRead(clock1);
```

Hodiny `clock1` jsou přečteny a čas v sekundách je uložen do proměnné `reg1`.

Příklad 2

```
reg1:=ClkRead(clock1 \HighRes);
```

Hodiny `clock1` jsou přečteny a čas v sekundách je uložen s vysokým rozlišením do proměnné `reg1`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Čas v sekundách uložený v hodinách. Rozlišení je normálně 0,001 sekundy. Při používání přepínače `HighRes` je možné získat rozlišení 0,000001 sekundy.

Argument

```
ClkRead (Clock \HighRes)
```

`Clock`

Datový typ: `clock`

Jméno hodin, které budou přečteny.

`[ \HighRes ]`

High Resolution

Datový typ: `switch`

Určuje, že čas by měl být čten s vyšším rozlišením. Jestliže se používá tento přepínač, je možné číst čas s rozlišením 0,000001.

Vzhledem k preciznosti datového typu num můžete získat pouze mikrosekundové rozlišení, dokud je čtená hodnota méně než 1 sekunda.

Vykonávání programu

Hodiny je možné číst, když jsou zastaveny nebo běží.

Jakmile jsou hodiny přečteny, je možné je číst znovu, znovu spustit, zastavit nebo resetovat.

Řešení chyb

Jestliže hodiny běží 4 294 967 sekund (49 dní 17 hodin 2 minuty 47 sekund), dojde k jejich přeplnění a systémová proměnná `ERRNO` je nastavena na `ERR_OVERFLOW`.

Chyba může být zpracována v handleru chyb.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.41 ClkRead - Čte hodiny použité pro časování

RobotWare-OS

Pokračování

Při používání přepínače `HighRes` **chyba** `ERR_OVERFLOW` **nemůže nastat, ale hodiny se otočí asi po 49 700 dnech.**

Syntaxe

```
ClkRead '('
  [ Clock ':=' ] < variable (VAR) of clock >
  [ '\' HighRes] ')'
```

Funkce s vrácenou hodnotou typu `num`.

Související informace

Pro informace o	Viz
Instrukce k hodinám	*Technická referenční příručka - Přehled RAPID*
Další příklady	*ClkReset - Spustí hodiny používané pro časování na str 121*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.42 CorrRead - Načítá aktuální celkové ofsety

Použití

`CorrRead` se používá ke čtení celkových korekcí dodaných všemi připojenými generátory korekcí.

`CorrRead` se může použít pro:

- zjištění, jak se aktuální dráha liší od původní dráhy.
- činnosti ke snížení rozdílu.

Základní příklady

Následující příklad názorně ukazuje funkci`CorrRead`.

Viz také *Další příklady na str 1107*.

Příklad 1

```
VAR pos offset;
...
offset := CorrRead();
```

Aktuální ofsety dodané všemi připojenými generátory korekcí jsou dostupné v proměnném ofsetu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `pos`

Celkové absolutní ofsety dosud dodané od všech připojených generátorů korekcí.

Další příklady

Více příkladů funkce `CorrRead` viz instrukci `CorrCon`.

Syntaxe

```
CorrRead '(' ')'
```

Funkce s vrácenou hodnotou datového typu `pos`.

Související informace

Pro informace o	Viz
Připojuje se ke generátoru korekcí	*CorrCon - Připojuje ke generátoru korekcí na str 140*
Odpojuje se od generátoru korekcí	*CorrDiscon - Odpojuje se od generátoru korekcí na str 145*
Zapisuje do generátoru korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
CorrClear - Odstraní všechny generátory korekcí	*CorrClear - Odstraní všechny generátory korekcí na str 139*
Popisovač korekcí	*corrdescr - Popisovač generátoru korekcí na str 1480*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.43 Cos - Vypočítává hodnotu kosinu

Použití

`Cos` (*Cosine*) se používá k výpočtu hodnoty kosinu z hodnoty úhlu na datových typech `num`.

Základní příklady

Následující příklad názorně ukazuje funkci`Cos`.

Příklad 1

```
VAR num angle;
VAR num value;
...
...
value := Cos(angle);
```

`value` získá hodnotu kosinu `angle`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Hodnota kosinu, rozsah = [-1, 1] .

Argumenty

```
Cos (Angle)
```

`Angle`

Datový typ: `num`

Hodnota úhlu vyjádřená ve stupních.

Syntaxe

```
Cos '('
  [Angle ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.44 CosDnum - Vypočítává hodnotu kosinu

Použití

`CosDnum` (*Cosine dnum*) se používá k výpočtu hodnoty kosinu z hodnoty úhlu na datových typech `dnum`.

Základní příklady

Následující příklad názorně ukazuje funkci`CosDnum`.

Příklad 1

```
VAR dnum angle;
VAR dnum value;
...
...
value := CosDnum(angle);
```

`value` získá hodnotu kosinu `angle`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Hodnota kosinu, rozsah = [-1, 1] .

Argumenty

```
CosDnum (Angle)
```

`Angle`

Datový typ: `dnum`

Hodnota úhlu vyjádřená ve stupních.

Syntaxe

```
CosDnum '('
  [Angle ':='] <expression (IN) of dnum> ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.45 CPos - Čte aktuální poziční data (pos)

Použití

`CPos` *(Current Position)* se používá pro čtení aktuální pozice robotu.

Tato funkce vrací hodnoty x, y a z TCP robotu jako data typu pos. Jestliže kompletní pozice robotu (`robtarget`) má být přečtena, potom použijte namísto toho funkci `CRobT`.

Základní příklady

Následující příklad názorně ukazuje funkci`CPos`.

Viz také *Další příklady na str 1111*.

Příklad 1

```
VAR pos pos1;

MoveL *, v500, fine \Inpos := inpos50, tool1;
pos1 := CPos(\Tool:=tool1 \WObj:=wobj0);
```

Aktuální pozice TCP robotu je uložena do proměnné `pos1`. Nástroj `tool1` a pracovní objekt `wobj0` jsou použity pro výpočet pozice.

Všimněte si, že robot stojí v klidu, předtím, než je pozice přečtena a vypočítána. Toho se dosáhne použitím stop bodu `fine` v rámci přesnosti pozice `inpos50` v předchozí instrukci pohybu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `pos`

Aktuální pozice (pos) robotu s x, y a z v nejvzdálenějším souřadném systému, když vezmeme v úvahu určený nástroj, pracovní objekt a aktivní souřadný systém ProgDisp.

Argumenty

```
CPos([\Tool] [\WObj])
```

`[\Tool]`

Datový typ: `tooldata`

Nástroj použitý pro výpočet aktuální pozice robotu.

Jestliže je tento argument vynechán, použije se aktuální aktivní nástroj.

`[\WObj]`

Work Object

Datový typ: `wobjdata`

Pracovní objekt (souřadný systém), ke kterému se vztahuje aktuální pozice robotu vrácená funkcí.

Jestliže je tento argument vynechán, použije se aktuální aktivní pracovní objekt.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

VAROVÁNÍ

Doporučuje se během programování vždy určit argumenty `\Tool` a `\WObj`. Funkce potom vždy vrátí požadovanou pozici, i když je aktivován jiný nástroj nebo pracovní objekt.

Vykonávání programu

Vrácené souřadnice reprezentují pozici TCP v souřadném systému ProgDisp.

Další příklady

Více příkladů funkce `CPos` je názorně uvedeno dole.

```
VAR pos pos2;
VAR pos pos3;
VAR pos pos4;

pos2 := CPos(\Tool:=grip3 \WObj:=fixture);
...
pos3 := CPos(\Tool:=grip3 \WObj:=fixture);
pos4 := pos3-pos2;
```

Pozice x, y a z robotu je zachycena na dvou místech v programu pomocí funkce `CPos`. Nástroj `grip3` a pracovní objekt `fixture` se používají pro výpočet pozice. Vzdálenosti x, y a z proběhlé mezi těmito pozicemi jsou potom vypočítány a uloženy do proměnné `pos4`.

Syntaxe

```
CPos '('
  ['\' Tool ':=' <persistent (PERS) of tooldata>]
  ['\' WObj ':=' <persistent (PERS) of wobjdata>] ')'
```

Funkce s vrácenou hodnotou datového typu `pos`.

Související informace

Pro informace o	Viz
Definice pozice	*pos - Pozice (pouze X, Y a Z) na str 1553*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Souřadný systém ProgDisp	*PDispOn - Aktivuje posun programu na str 477*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Čtení aktuálního `robtarget`	*CRobT - Čte aktuální poziční data (robtarget) na str 1112*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.46 CRobT - Čte aktuální poziční data (robtarget)

Použití

`CRobT` (*(Current Robot Target)*) se používá pro čtení aktuální pozice robotu a externích os.

Tato funkce vrací hodnotu `robtarget` s pozicí (x, y, z), orientací (q1 ...q4), konfigurací os robotu a pozicí externích os. Jestliže se mají číst pouze hodnoty x, y a z TCP robotu (`pos`), použijte namísto toho funkci `CPos`.

Základní příklady

Následující příklad názorně ukazuje funkci`CRobT`.

Viz také *Další příklady na str 1113*.

Příklad 1

```
VAR robtarget p1;
MoveL *, v500, fine \Inpos := inpos50, tool1;
p1 := CRobT(\Tool:=tool1 \WObj:=wobj0);
```

Aktuální pozice robotu a externích os je uložena do `p1`. Nástroj `tool1` a pracovní objekt `wobj0` jsou použity pro výpočet pozice.

Všimněte si, že robot stojí v klidu, předtím, než je pozice přečtena a vypočítána. Toho se dosáhne použitím stop bodu `fine` v rámci přesnosti pozice `inpos50` v předchozí instrukci pohybu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `robtarget`

Aktuální pozice robotu a externích os v nejvzdálenějším souřadném systému, když vezmeme v úvahu určený nástroj, pracovní objekt a aktivní souřadný systém `ProgDisp/ExtOffs`.

Argumenty

```
CRobT ([\TaskRef]|[\TaskName] [\Tool] [\WObj])
```

`[\TaskRef]`

Task Reference

Datový typ: `taskid`

Identita programové úlohy, ze které by měl být přečten `robtarget`.

Pro všechny programové úlohy v systému budou dostupné předdefinované proměnné datového typu `taskid`. Identita proměnné bude "taskname"+"Id", například, pro úlohu `T_ROB1` bude identita proměnné `T_ROB1Id`.

`[\TaskName]`

Datový typ: `string`

Jméno programové úlohy, ze které by měl být přečten `robtarget`.

Jestliže žádný z argumentů `\TaskRef` or `\TaskName` není určen, potom se použije aktuální úloha.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[\Tool]`

Datový typ: `tooldata`

Perzistentní proměnná pro nástroj použitý k výpočtu aktuální pozice robotu.

Jestliže je tento argument vynechán, použije se aktuální aktivní nástroj.

`[\WObj]`

Work Object

Datový typ: `wobjdata`

Perzistentní proměnná pro pracovní objekt (souřadný systém), ke kterému se vztahuje aktuální pozice robotu vrácená funkcí.

Jestliže je tento argument vynechán, použije se aktuální aktivní pracovní objekt.

VAROVÁNÍ

Doporučuje se během programování vždy určit argumenty `\Tool` a `\WObj`. Funkce potom vždy vrátí požadovanou pozici, i když je aktivován jiný nástroj nebo pracovní objekt.

Vykonávání programu

Vrácené souřadnice reprezentují pozici TCP v souřadném systému `ProgDisp`. Externí osy jsou reprezentovány v souřadném systému `ExtOffs`.

Jestliže je použit jeden z argumentů `\TaskRef` nebo `\TaskName`, ale argumenty `Tool` a `WObj` nejsou použity, potom bude použit aktuální nástroj a pracovní objekt v určené úloze.

Další příklady

Více příkladů funkce `CRobT` je názorně uvedeno dole.

Příklad 1

```
VAR robtarget p2;
p2 := ORobT( CRobT(\Tool:=grip3 \WObj:=fixture) );
```

Aktuální pozice v souřadném systému objektu (bez `ProgDisp` nebo `ExtOffs`) robotu a externích os je uložena do `p2`. Nástroj `grip3` a pracovní objekt `fixture` jsou použity pro výpočet pozice.

Příklad 2

```
! In task T_ROB1
VAR robtarget p3;
p3 := CRobT(\TaskRef:=T_ROB2Id \Tool:=tool1 \WObj:=wobj0);
```

Aktuální pozice robotu a externích os v úloze `T_ROB2` je uložena do `p3` v úloze `T_ROB1`. Nástroj `tool1` a pracovní objekt `wobj0` jsou použity pro výpočet pozice.

Všimněte si, že robot v úloze `T_ROB2`se může pohybovat, když je pozice čtena a vypočítávána. Aby bylo zajištěno, že robot bude stát v klidu, může být naprogramován stop bod `fine` v předchozí pohybové instrukci v úloze `T_ROB2`, a instrukce `WaitSyncTask` může být použita k synchronizaci instrukcí v úloze `T_ROB1`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.46 CRobT - Čte aktuální poziční data (robtarget)

RobotWare - OS

Pokračování

Příklad 3

```
! In task T_ROB1
VAR robtarget p4;
p4 := CRobT(\TaskName:="T_ROB2");
```

Aktuální pozice robotu a externích os v úloze `T_ROB2` je uložena do `p4` v úloze `T_ROB1`. Aktuální nástroj a pracovní objekt `T_ROB2` jsou použity pro výpočet pozice.

Řešení chyb

Jestliže argument `\TaskRef` nebo `\TaskName` určuje některou nepohybovou úlohu, potom je systémová proměnná `ERRNO` nastavena na `ERR_NOT_MOVETASK`. Tato chyba může být ošetřena v chybovém handleru.

Ale žádná chyba nebude generována, jestliže argumenty `\TaskRef` nebo `\TaskName` určují nepohybovou úlohu, která vykonává tuto funkci `CRobT` (reference k mé vlastní nepohybové úloze). Pozice bude potom získána od připojené pohybové úlohy.

Syntaxe

```
CRobT '('
  ['\' TaskRef ':=' <variable (VAR) of taskid>]
  |['\' TaskName ':=' <expression (IN) of string>]
  ['\'Tool ':=' <persistent (PERS) of tooldata>]
  ['\' WObj ':=' <persistent (PERS) of wobjdata>] ')'
```

Funkce s vrácenou hodnotou datového typu `robtarget`.

Související informace

Pro informace o	Viz
Definice pozice	*robtarget - Poziční data na str 1572*
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Souřadný systém ProgDisp	*PDispOn - Aktivuje posun programu na str 477*
Souřadný systém ExtOffs	*EOffsOn - Aktivuje ofset pro pomocné osy na str 197*
Čtení aktuálního `pos` (pouze x, y, z)	*CPos - Čte aktuální poziční data (pos) na str 1110*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.47 CSpeedOverride - Čte aktuální rychlost potlačení

Použití

`CSpeedOverride` se používá pro čtení potlačení rychlosti nastavené operátorem z FlexPendantu. Vrácená hodnota je zobrazena jako procentuální část, kde 100 % odpovídá naprogramované rychlosti.

V aplikacích s instrukcí `SpeedRefresh` může být tato funkce také použita ke čtení aktuální hodnoty potlačení rychlosti pro tuto nebo připojení pohybové programové úlohy.

Zapamatujte si! Nesmí být smíšeno s argumentem `Override` v instrukci RAPID `VelSet`.

Základní příklady

Následující příklad názorně ukazuje funkci`CSpeedOverride`.

Příklad 1

```
VAR num myspeed;
myspeed := CSpeedOverride();
```

Aktuální potlačení rychlosti bude uloženo do proměnné `myspeed`. Například, jestliže hodnota je 100, potom je to ekvivalent ke 100 %.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Hodnota potlačení rychlosti v procentech naprogramované rychlosti. Toto bude numerická hodnota v rozsahu 0 - 100.

Argumenty

```
CSpeedOverride ( [\CTask] )
```

`[\CTask]`

Datový typ: `switch`

Získejte aktuální hodnotu potlačení rychlosti pro tuto nebo připojenou pohybovou programovou úlohu. Používejte společně s instrukcí `SpeedRefresh`.

Jestliže se tento argument nepoužije, potom funkce vrací aktuální potlačení rychlosti pro celý systém (všechny pohybové programové úlohy). Znamená to ruční potlačení rychlosti nastavené z FlexPendantu.

Syntaxe

```
CSpeedOverride '('
  ['\' CTask ] ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Změna rychlosti potlačení	*Návod k použití - IRC5 s jednotkou FlexPendant*, sekce *Programování a testování průběhu produkce - Nabídka rychlého nastavení, Rychlost*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.47 CSpeedOverride - Čte aktuální rychlost potlačení

RobotWare - OS

Pokračování

Pro informace o	Viz
Aktualizovat potlačení rychlosti z RAPI-Du	*SpeedRefresh - Aktualizovat potlačení rychlosti pro probíhající pohyb na str 697*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.48 CTime - Čte přesný čas jako řetězec

Použití

`CTime` () se používá pro čtení aktuálního systémového času.

Tuto funkci je možné používat k předložení aktuálního času operátorovi na displeji FlexPendantu nebo k poslání aktuálního času do textového souboru, do kterého program zapisuje.

Základní příklady

Následující příklad názorně ukazuje funkci`CTime`.

Viz také *Další příklady na str 1117*.

Příklad 1

```
VAR string time;
time := CTime();
```

Aktuální čas je uložen do proměnné `time`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Aktuální čas v řetězci.

Standardní formát času je “hodiny:minuty:sekundy”, například "18:20:46".

Další příklady

Více příkladů funkce `CTime` je názorně uvedeno dole.

Příklad 1

```
VAR string time;
time := CTime();
TPWrite "The current time is: "+time;
Write logfile, time;
```

Aktuální čas je zapsán na displej FlexPendantu a do textového souboru.

Syntaxe

```
CTime '(' ')'
```

Funkce s vrácenou hodnotou typu `string`.

Související informace

Pro informace o	Viz
Instrukce k datumu a času	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Systémový čas*
Nastavování systémových hodin	*Návod k použití - IRC5 s jednotkou FlexPendant*, sekce *Změna nastavení FlexPendant*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.49 CTool - Čte data aktuálního nástroje

Použití

`CTool` ***Current Tool*** se používá pro čtení dat aktuálního nástroje.

Základní příklady

Následující příklad názorně ukazuje funkci `CTool`.

Příklad 1

```
PERS tooldata temp_tool:= [ TRUE, [ [0, 0, 0], [1, 0, 0 ,0] ],
     [0.001, [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0] ];
temp_tool := CTool();
```

Hodnota aktuálního nástroje je uložena do proměnné `temp_tool`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `tooldata`

Tato funkce vrací hodnotu `tooldata`, držící hodnotu aktuálního nástroje, tj. naposledy použitého nástroje v pohybové instrukci.

Vrácená hodnota reprezentuje pozici TCT a orientaci v souřadném systému středu zápěstí. Viz `tooldata`.

Argumenty

```
CTool ([\TaskRef]|[\TaskName])
```

`[\TaskRef]`

Task Reference

Datový typ: `taskid`

Identita programové úlohy, ze které by se měla načíst data aktuálního nástroje.

Pro všechny programové úlohy v systému budou dostupné předdefinované proměnné datového typu `taskid`. Identita proměnné bude "taskname"+"Id", například, pro úlohu `T_ROB1` bude identita proměnné `T_ROB1Id`.

`[\TaskName]`

Datový typ: `string`

Jméno programové úlohy, ze které by se měla načíst data aktuálního nástroje.

Jestliže žádný z argumentů `\TaskRef` or `\TaskName` není určen, potom se použije aktuální úloha.

Řešení chyb

Jestliže argument `\TaskRef` nebo `\TaskName` určuje některou nepohybovou úlohu, potom je systémová proměnná `ERRNO` nastavena na `ERR_NOT_MOVETASK`. Tato chyba může být ošetřena v chybovém handleru.

Ale žádná chyba nebude generována, jestliže argumenty `\TaskRef` nebo `\TaskName` určují nepohybovou úlohu, která vykonává tuto funkci `CTool` (reference k mé vlastní nepohybové úloze). Data nástroje budou potom získána od připojené pohybové úlohy.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
CTool '('
  ['\' TaskRef ':=' <variable (VAR) of taskid>]
  |['\' TaskName ':=' <expression (IN) of string>] ')'
```

Funkce s vrácenou hodnotou datového typu `tooldata`.

Související informace

Pro informace o	Viz
Definice nástrojů	*tooldata - Data nástroje na str 1610*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Souřadné systémy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.50 CWObj - Čte aktuální data pracovního objektu

Použití

CWObj *Current Work Object* se používá pro čtení dat aktuálního pracovního objektu.

Základní příklady

Následující příklad názorně ukazuje funkci CWObj.

Příklad 1

```
PERS wobjdata temp_wobj:= [FALSE, TRUE, "", [[0,0,0], [1,0,0,0]],
     [[0,0,0], [1,0,0,0]]];
temp_wobj := CWObj();
```

Hodnota aktuálního pracovního objektu je uložena do proměnné temp_wobj.

Vratná hodnota (Vrátit hodnotu)

Datový typ: wobjdata

Tato funkce vrací hodnotu wobjdata, držící hodnotu aktuálního pracovního objektu, tj. naposledy použitého pracovního objektu v pohybové instrukci.

Vrácená hodnota reprezentuje pozici pracovního objektu a orientaci ve světovém souřadném systému. Viz wobjdata.

Argumenty

```
CWobj ([\TaskRef]|[\TaskName])
```

[\TaskRef]

Task Reference

Datový typ: taskid

Identita programové úlohy, ze které by se měla načíst data aktuálního pracovního objektu.

Pro všechny programové úlohy v systému budou dostupné předdefinované proměnné datového typu taskid. Identita proměnné bude "taskname"+"Id", například, pro úlohu T_ROB1 bude identita proměnné T_ROB1Id.

[\TaskName]

Datový typ: string

Jméno programové úlohy, ze které by se měla načíst data aktuálního pracovního objektu.

Jestliže žádný z argumentů \TaskRef or \TaskName není určen, potom se použije aktuální úloha.

Řešení chyb

Jestliže argument \TaskRef nebo \TaskName určuje některou nepohybovou úlohu, potom je systémová proměnná ERRNO nastavena na ERR_NOT_MOVETASK. Tato chyba může být ošetřena v chybovém handleru.

Ale žádná chyba nebude generována, jestliže argumenty \TaskRef nebo \TaskName určují nepohybovou úlohu, která vykonává tuto funkci CWobj (reference

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

k mé vlastní nepohybové úloze). Data pracovního nástroje budou potom získána od připojené pohybové úlohy.

Syntaxe

```
CWobj '('
  ['\' TaskRef ':=' <variable (VAR) of taskid>]
  |['\' TaskName ':=' <expression (IN) of string>] ')'
```

Funkce s vrácenou hodnotou datového typu `wobjdata`.

Související informace

Pro informace o	Viz
Definice pracovních objektů	*wobjdata - Data pracovního objektu na str 1634*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Souřadné systémy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.51 DecToHex - Převést z desítkového na šestnáctkový formát

Použití

`DecToHex` se používá k převodu čísla určeného v čitelném řetězci v základně 10 do základny 16.

Výsledný řetězec je konstruován ze znakové sady [0-9,A-F,a-f].

Tato rutina zpracovává čísla od 0 do 9223372036854775807dec nebo 7FFFFFFFFFFFFFFF hex.

Základní příklady

Následující příklad názorně ukazuje funkci`DecToHex`.

Příklad 1

```
VAR string str;

str := DecToHex("99999999");
```

Proměnné `str` je dána hodnota `"5F5E0FF"`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Řetězec převedený do šestnáctkové reprezentace daného čísla v inparametrovém řetězci.

Argumenty

```
DecToHex ( Str )
```

`Str`

String

Datový typ: `string`

Řetězec k převedení.

Syntaxe

```
DecToHex '('
  [ Str ':=' ] <expression (IN) of string> ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Řetězcové funkce*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Základní prvky*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.52 DefAccFrame - Definovat přesný rámec

Použití

`DefAccFrame` (*Define Accurate Frame*) se používá pro definování rámce od tří do deseti původních pozic a stejného počtu posunutých pozic.

Popis

Rámec může být definován, když sada cílů je známá na dvou odlišných místech. Tudíž, *stejné fyzické pozice* jsou použity, ale vyjádřeny jsou odlišně.

Posuďte to ze dvou odlišných přístupů:

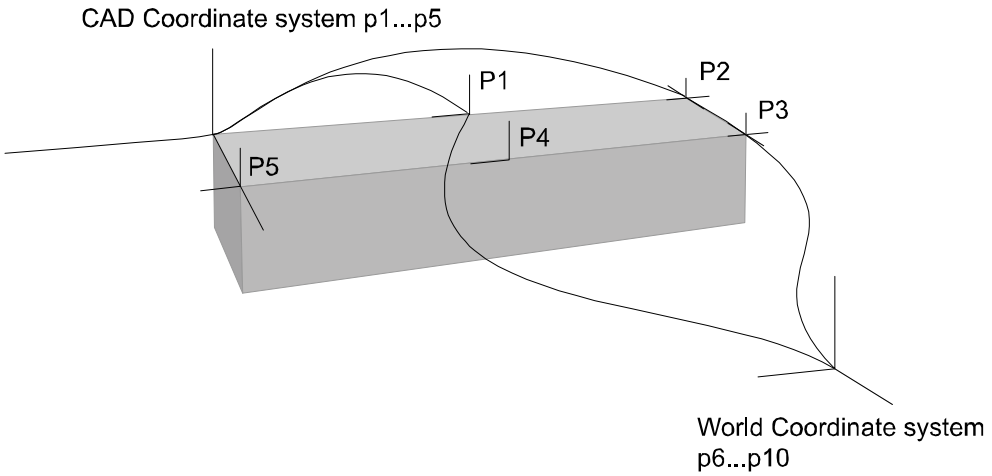
1. Stejné fyzické pozice jsou vyjádřeny ve vztahu k různým souřadným systémům. Například, řada pozic je získána z výkresu CAD, tudíž tyto pozice jsou vyjádřeny v lokálním souřadném systému CAD. Stejné pozice jsou potom vyjádřeny ve světovém souřadném systému robotu. Z těchto dvou sad pozic je vypočítán rámec mezi souřadným systémem CAD a světovým souřadným systémem robotu.
2. Řada pozic se vztahuje k objektu v původní pozici. Po posunu objektu jsou pozice determinovány znovu (po vyhledání). Z těchto dvou sad pozic (staré pozice, nové pozice) je vypočítán rámec posunu.

Tři cíle stačí pro definování rámce, ale kvůli zvýšení přesnosti by mělo být použito několik bodů.

Základní příklady

Následující příklad názorně ukazuje funkci`DefAccFrame`.

Příklad 1



xx0500002179

```
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];
CONST robtarget p4 := [...];
CONST robtarget p5 := [...];
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.52 DefAccFrame - Definovat přesný rámec

RobotWare - OS

Pokračování

```
VAR robtarget p6 := [...];
VAR robtarget p7 := [...];
VAR robtarget p8 := [...];
VAR robtarget p9 := [...];
VAR robtarget p10 := [...];
VAR robtarget pWCS{5};
VAR robtarget pCAD{5};

VAR pose frame1;
VAR num max_err;
VAR num mean_err;

! Add positions to robtarget arrays
pCAD{1}:=p1;
...
pCAD{5}:=p5;

pWCS{1}:=p6;
...
pWCS{5}:=p10;
frame1 := DefAccFrame (pCAD, pWCS, 5, max_err, mean_err);
```

Pět pozic `p1- p5` **souvisejících s objektem bylo uloženo. Pět pozic je také uloženo ve vztahu ke světovému souřadnému systému jako** `p6-p10`**. Z těchto 10 pozic je vypočítán rámec** `frame1` **mezi objektem a světovým souřadným systémem. Rámcem bude CAD rámec vyjádřený ve světovém souřadném systému. Jestliže se změní vstupní pořadí seznamů cílů, tj.** `DefAccFrame` **(**`pWCS, pCAD...`**), potom bude světový rámec vyjádřen v souřadném systému CAD.**

Vratná hodnota (Vrátit hodnotu)

Datový typ: `pose`

Vypočítaný rámec `TargetListOne` **vyjádřený v souřadném systému** `TargetListTwo`

Argumenty

```
DefAccFrame (TargetListOne TargetListTwo TargetsInList
     MaxErrMeanErr)
```

`TargetListOne`

Datový typ: `robtarget`

Pole robtarget držící pozice definované v souřadném systému jedna. Min počet robtarget je 3, maximum je 10.

`TargetListTwo`

Datový typ: `robtarget`

Pole robtarget držící pozice definované v souřadném systému dvě. Min počet robtarget je 3, maximum je 10.

`TargetsInList`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Počet robtarget v poli.

`MaxErr`

Datový typ: `num`

Odhadovaná maximální chyba v mm.

`MeanErr`

Datový typ: `num`

Odhadovaná střední chyba v mm.

Řešení chyb

Jestliže pozice nemají požadovaný vztah nebo nejsou určeny s dostatečnou přesností, potom je systémová proměnná `ERRNO` je nastavena na `ERR_FRAME` . Tuto chybu je možné potom ošetřit v chybovém handleru.

Syntaxe

```
DefAccFrame '('
  [TargetListOne ':='] <array {*} (IN) of robtarget> ','
  [TargetListTwo ':='] <array {*} (IN) of robtarget> ','
  [TargetsInList ':='] <expression (IN) of num> ','
  [MaxErr ':='] <variable (VAR) of num> ','
  [MeanErr ':='] <variable (VAR) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `pose`.

Související informace

Pro informace o	Viz
Výpočet rámce ze tří pozic	*DefFrame - Definovat rámec na str 1129*
Výpočet rámce ze šesti pozic	*DefDFrame - Definovat rámec posunu na str 1126*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.53 DefDFrame - Definovat rámec posunu

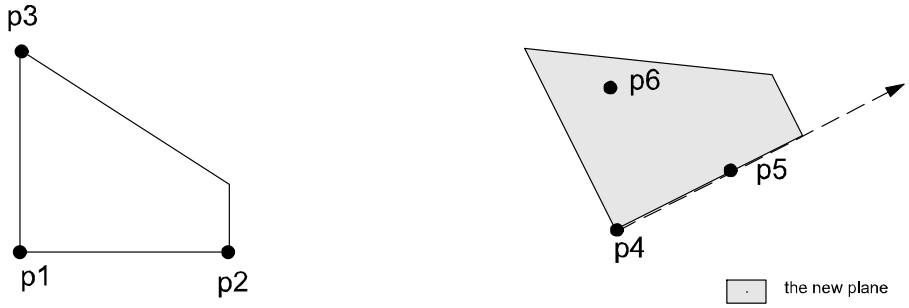
Použití

`DefDFrame`(***Define Displacement Frame***) se používá k výpočtu rámce posunu ze tří původních pozic a tří posunutých pozic.

Základní příklady

Následující příklad názorně ukazuje funkci`DefDFrame`.

Příklad 1



xx0500002177

```
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];
VAR robtarget p4;
VAR robtarget p5;
VAR robtarget p6;
VAR pose frame1;
...
!Search for the new positions
SearchL sen1, p4, *, v50, tool1;
...
SearchL sen1, p5, *, v50, tool1;
...
SearchL sen1, p6, *, v50, tool1;
frame1 := DefDframe (p1, p2, p3, p4, p5, p6);
...
!Activation of the displacement defined by frame1
PDispSet frame1;
```

Tři pozice `p1-p3` související s objektem v původní pozici byly uloženy. Po posunu objektu jsou vyhledány tři nové pozice a uloženy jako `p4-p6`. Rámec posunu je vypočítán z těchto šesti pozic. Potom je vypočítaný rámec použit k posunu všech uložených pozic v programu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `pose`

Rámec posunu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Argumenty

```
DefDFrame (OldP1 OldP2 OldP3 NewP1 NewP2 NewP3)
```

`OldP1`

Datový typ: `robtarget`

První původní pozice.

`OldP2`

Datový typ: `robtarget`

Druhá původní pozice.

`OldP3`

Datový typ: `robtarget`

Třetí původní pozice.

`NewP1`

Datový typ: `robtarget`

První posunutá pozice. Rozdíl mezi `OldP1` **a** `NewP1` **bude definovat překladovou část rámce a musí být změřen a určen s velkou přesností.**

`NewP2`

Datový typ: `robtarget`

Druhá posunutá pozice. Linka `NewP1 ... NewP2` **bude definovat rotaci staré linky** `OldP1 ... OldP2`**.**

`NewP3`

Datový typ: `robtarget`

Třetí posunutá pozice. Tato pozice bude definovat rotaci roviny, například, měla by být umístěna na novou rovinu `NewP1`**,** `NewP2`**, a** `NewP3`**.**

Řešení chyb

Jestliže není možné vypočítat rámec kvůli špatné přesnosti v pozicích, potom je systémová proměnná `ERRNO` **nastavena na** `ERR_FRAME`**. Tato chyba může být potom ošetřena v chybovém handleru.**

Syntaxe

```
DefDFrame '('
  [OldP1 ':='] <expression (IN) of robtarget> ','
  [OldP2 ':='] <expression (IN) of robtarget> ','
  [OldP3 ':='] <expression (IN) of robtarget> ','
  [NewP1 ':='] <expression (IN) of robtarget> ','
  [NewP2 ':='] <expression (IN) of robtarget> ','
  [NewP3 ':='] <expression (IN) of robtarget> ')'
```

Funkce s vrácenou hodnotou datového typu `pose`**.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.53 DefDFrame - Definovat rámec posunu

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Aktivace rámce posunu	*PDispSet - Aktivuje posun programu pomocí známého rámce na str 481*
Ruční definice rámce posunu	*Návod k použití - IRC5 s jednotkou FlexPendant*, sekce *Kalibrování*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.54 DefFrame - Definovat rámec

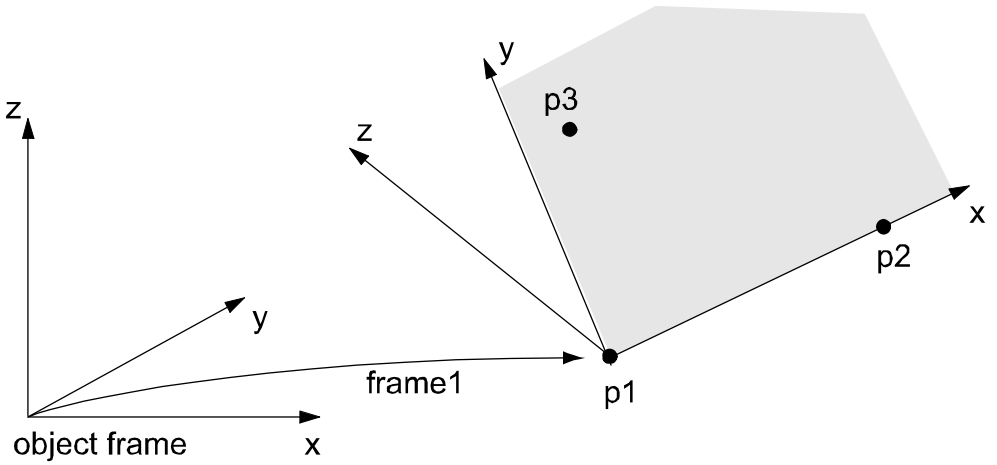
Použití

`DefFrame` (*Define Frame*) se používá k výpočtu rámce ze tří pozic definujících rámec.

Základní příklady

Následující příklad názorně ukazuje funkci`DefFrame`.

Příklad 1



xx0500002181

Tři pozice `p1- p3` související se souřadným systémem objektu jsou použity k definování nového souřadného systému, `frame1`. První pozice `p1` definuje počátek nového souřadného systému. Druhá pozice `p2` definuje směr osy x. Třetí pozice `p3` definuje umístění roviny xy. Definovaný `frame1` se může používat jako rámec posunu, jak je vidět na příkladu dole:

```
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];
VAR pose frame1;
...
...
frame1 := DefFrame (p1, p2, p3);
...
...
!Activation of the displacement defined by frame1
PDispSet frame1;
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `pose`

Vypočítaný rámec.

Výpočet souvisí se souřadným systémem aktivního objektu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.54 DefFrame - Definovat rámec

RobotWare - OS

Pokračování

Argumenty

```
DefFrame (NewP1 NewP2 NewP3 [\Origin])
```

`NewP1`

Datový typ: `robtarget`

První pozice, která bude definovat počátek nového souřadného systému.

`NewP2`

Datový typ: `robtarget`

Druhá pozice, která bude definovat směr osy x nového souřadného systému.

`NewP3`

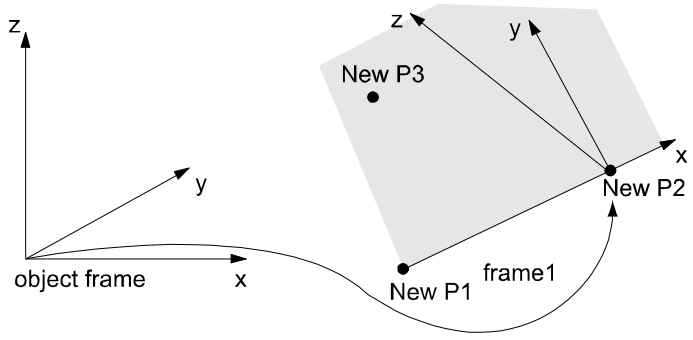
Datový typ: `robtarget`

Třetí pozice, která bude definovat rovinu xy nového souřadného systému. Pozice bodu 3 bude na kladné straně y, viz obrázek nahoře.

`[\Origin]`

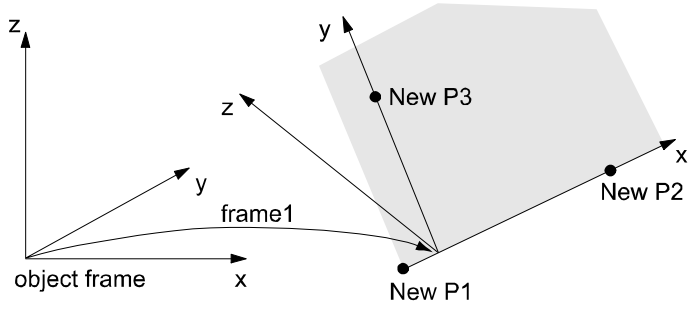
Datový typ: `num`

Volitelný argument, který bude definovat, jak bude umístěn počátek nového souřadného systému. `Origin` **= 1 znamená, že počátek je umístěn do** `NewP1`**, tj. stejně, jako kdyby tento argument byl vynechán.** `Origin` **= 2 znamená, že počátek je umístěn do** `NewP2`**. Viz obrázek dole.**



xx0500002178

`Origin` **= 3 znamená, že počátek je umístěn na řádku procházející** `NewP1` **a** `NewP2`**, takže** `NewP3` **bude umístěn na osu y. Viz obrázek dole.**



xx0500002180

Ostatní hodnoty, nebo jestliže `Origin` **je vynechán, umístí počátek do** `NewP1`**.**

Pokračování na další straně

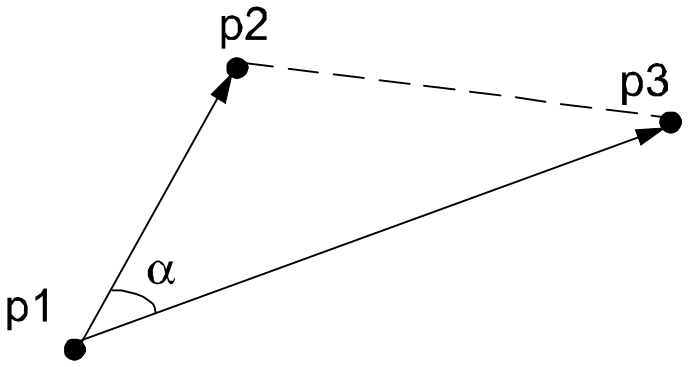
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Jestliže není možné vypočítat rámec kvůli dolním omezením, potom je systémová proměnná ERRNO nastavena na ERR_FRAME. Tato chyba může být potom ošetřena v chybovém handleru.

Omezení

Tři pozice p1 - p3, definující rámec, musí definovat dobře tvarovaný trojúhelník. Nejlépe tvarovaný trojúhelník je takový, který má všechny strany stejně dlouhé.



xx0500002182

Trojúhelník není považován za dobře tvarovaný, jestliže úhel α je příliš malý. Úhel α je příliš malý, když:

$|\cos \alpha| < 1 - 10^{-4}$

Trojúhelník p1, p2, p3 nesmí být příliš malý, tj. pozice nesmí být příliš blízko. Vzdálenosti mezi pozicemi p1 - p2 a p1 - p3 nesmí být méně než 0,1 mm.

Syntaxe

```
DefFrame '('
  [NewP1 ':='] <expression (IN) of robtarget> ','
  [NewP2 ':='] <expression (IN) of robtarget> ','
  [NewP3 ':='] <expression (IN) of robtarget>
  ['\' Origin ':=' <expression (IN) of num >] ')'
```

Funkce s vrácenou hodnotou datového typu pose.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Matematika*
Aktivace rámce posunu	*PDispSet - Aktivuje posun programu pomocí známého rámce na str 481*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.55 Dim - Získává velikost pole

Použití

`Dim` (*Dimension*) se používá k získání jistého počtu elementů v poli.

Základní příklady

Následující příklad názorně ukazuje funkci`Dim`.

Viz také *Další příklady na str 1132*.

Příklad 1

```
PROC arrmul(VAR num array{*}, num factor)
  FOR index FROM 1 TO Dim(array, 1) DO
    array{index} := array{index} * factor;
  ENDFOR
ENDPROC
```

Všechny elementy num pole jsou násobeny faktorem. Tato procedura může pojmout každé jednorozměrové pole datového typu `num` jako vstup.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Počet elementů pole určeného rozměru.

Argumenty

```
Dim (ArrPar DimNo)
```

`ArrPar`

Array Parameter

Datový typ: Jakýkoliv typ

Jméno pole.

`DimNo`

Dimension Number

Datový typ: `num`

Požadovaný rozměr pole:

1 = první rozměr

2 = druhý rozměr

3 = třetí rozměr

Další příklady

Více příkladů jak používat instrukci `Dim` je názorně uvedeno dole.

Příklad 1

```
PROC add_matrix(VAR num array1{*,*,*}, num array2{*,*,*})

  IF Dim(array1,1) <> Dim(array2,1) OR Dim(array1,2) <>
       Dim(array2,2) OR Dim(array1,3) <> Dim(array2,3) THEN
    TPWrite "The size of the matrices are not the same";
    Stop;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
    ELSE
      FOR i1 FROM 1 TO Dim(array1, 1) DO
        FOR i2 FROM 1 TO Dim(array1, 2) DO
          FOR i3 FROM 1 TO Dim(array1, 3) DO
            array1{i1,i2,i3} := array1{i1,i2,i3} + array2{i1,i2,i3};
          ENDFOR
        ENDFOR
      ENDFOR
    ENDIF
    RETURN;

  ENDPROC
```

Jsou přidány dvě matrice. Jestliže se velikost matric liší, potom program zastaví a objeví se chybová zpráva.

Tato procedura může pojmout jakékoliv třírozměrné pole datového typu `num` **jako vstup.**

Syntaxe

```
Dim '('
  [ArrPar ':='] <reference (REF) of any type> ','
  [DimNo ':='] <expression (IN) of num> ')'
```

Parametr REF vyžaduje, aby odpovídající argument byl buď konstanta, proměnná nebo celý perzistent. Argument by mohl být také parametr IN, parametr VAR nebo celý parametr PERS.

Funkce s vrácenou hodnotou datového typu `num`**.**

Související informace

Pro informace o	Viz
Parametry pole	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Rutiny*
Deklarace pole	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Data*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.56 DInput - Čte hodnotu digitálního vstupního signálu

Použití

`DInput` se používá pro čtení aktuální hodnoty digitálního vstupního signálu.

> **POZNÁMKA**
>
> Všimněte si, že funkce `DInput` je zděděná funkce, kterou již není nutné používat. Viz příklady alternativních a doporučených způsobů programování.

Základní příklady

Následující příklad názorně ukazuje funkci`DInput`.

Viz také *Další příklady na str 1134*.

Příklad 1

```
IF DInput(di2) = 1 THEN ...
...
IF di2 = 1 THEN ...
```

Jestliže aktuální hodnota signálu `di2` je rovna 1, potom ...

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Aktuální hodnota signálu (0 nebo 1).

Argumenty

```
DInput (Signal)
```

`Signal`

Datový typ: `signaldi`

Jméno digitálního vstupu, které má být přečteno.

Vykonávání programu

Čtená hodnota signálu závisí na konfiguraci signálu. Jestliže je signál invertován v systémových parametrech, vrácená hodnota touto funkcí je opačná k hodnotě fyzického kanálu.

Další příklady

Více příkladů jak používat instrukci `DInput` je názorně uvedeno dole.

Příklad 1

```
weld_flag := DInput(weld);
...
weld_flag := weld;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Proměnná `weld_flag` se nastaví na stejnou hodnotu, jako je aktuální hodnota signálu`weld`.

POZNÁMKA

Všimněte si, v tomto případě `weld_flag` reflektuje aktuální hodnotu signálu. Tudíž, jestliže `weld_flag` je použit později v programu, nemůžete mít jistotu, že bude reflektovat aktuální hodnotu signálu.

Syntaxe

```
DInput '('
  [ Signal ':=' ] < variable (VAR) of signaladi > ')'
```

Funkce s vrácenou hodnotou datového typu `dionum`.

Související informace

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.57 Distance - Vzdálenost mezi dvěma body.

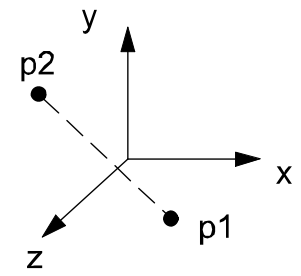
Použití

`Distance` se používá pro výpočet vzdálenosti mezi dvěma body v prostoru.

Základní příklady

Následující příklad názorně ukazuje funkci`Distance`.

Příklad 1



xx0500002321

```
VAR num dist;
CONST pos p1 := [4,0,4];
CONST pos p2 := [-4,4,4];
...
dist := Distance(p1, p2);
```

Vzdálenost v prostoru mezi dvěma body `p1` a `p2` je vypočítána a uložena do proměnné `dist`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Vzdálenost (vždy kladná) v mm mezi body.

Argumenty

```
Distance (Point1 Point2)
```

`Point1`

Datový typ: `pos`

První bod popsaný datovým typem `pos`.

`Point2`

Datový typ: `pos`

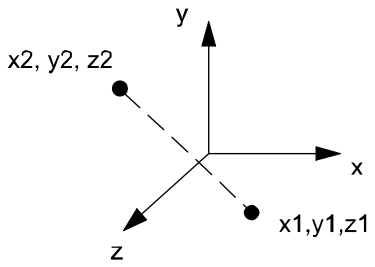
Druhý bod popsaný datovým typem `pos`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Výpočet vzdálenosti mezi dvěma body:



xx0500002322

$$\text{distance} = \sqrt{\left((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2\right)}$$

xx0500002323

Syntaxe

```
Distance '('
[Point1 ':='] <expression (IN) of pos> ','
[Point2 ':='] <expression (IN) of pos> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Matematika*
Definice pos	*pos - Pozice (pouze X, Y a Z) na str 1553*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.58 DIV - Vyhodnocuje dělení celého čísla

Použití

`DIV` je podmíněný výraz, který se používá k vyhodnocení dělení celých čísel.

Základní příklady

Následující příklady názorně ukazují funkci`DIV`.

Příklad 1

```
reg1 := 14 DIV 4;
```

Vrácená hodnota je `3` protože`14` může být děleno `4` `3`krát.

Příklad 2

```
VAR dnum mydnum1 := 10;
VAR dnum mydnum2 := 5;
VAR dnum mydnum3;
...
mydnum3 := mydnum1 DIV mydnum2;
```

Vrácená hodnota je `2` protože `10` může být děleno `5` `2` krát.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `numdnum`

Vrací celé číslo (integer), celé číslo, z dělení celých čísel (integer).

Syntaxe

```
<expression of num> DIV <expression of num>
```

Funkce s vrácenou hodnotou datového typu `num`.

```
<expression of dnum> DIV <expression of dnum>
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
num - Numerické hodnoty	*num - Numerické hodnoty na str 1538*
dnum - Dvojité numerické hodnoty	*dnum - Dvojité numerické hodnoty na str 1485*
MOD	*MOD - Vyhodnocuje modulo celého čísla na str 1225*
Výrazy	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.59 DnumToNum - Převádí dnum na num

Použití

DnumToNum převádí dnum na num, pokud je to možné, jinak generuje odstranitelnou chybu.

Základní příklady

Následující příklad názorně ukazuje funkciDnumToNum.

Příklad 1

```
VAR num mynum:=0;
VAR dnum mydnum:=8388607;
VAR dnum testFloat:=8388609;
VAR dnum anotherdnum:=4294967295;
! Works OK
mynum:=DnumToNum(mydnum);
! Accept floating point value
mynum:=DnumToNum(testFloat);
! Cause error recovery error
mynum:=DnumToNum(anotherdnum \Integer);
```

Dnum hodnota 8388607 je vrácena funkcí jako num hodnota 8388607.

Dnum hodnota 8388609 je vrácena funkcí jako num hodnota 8.38861E+06.

Dnum hodnota 4294967295 generuje odstranitelnou chybu ERR_ARGVALERR.

Vratná hodnota (Vrátit hodnotu)

Datový typ: num

Vstupní dnum hodnota může být v rozsahu -8388607 až 8388608 a vrací stejnou hodnotu jako num. Jestliže není použit přepínač \Integer, vstupní hodnota dnum může být v rozsahu -3.40282347E+38 až 3.40282347E+38 a vrácená hodnota se může stát hodnotou plovoucího bodu.

Argumenty

```
DnumToNum (Value [\Integer])
```

Value

Datový typ: dnum

Numerická hodnota, která bude převedena.

[\Integer]

Datový typ: switch

Pouze hodnoty celého čísla

Jestliže se nepoužívá spínač \Integer, je proveden „down cast“, i když hodnota se stává hodnotou plovoucího bodu. Jestliže není použit, je provedena kontrola, jestli je hodnotou celé číslo mezi -8388607 až 8388608. Jestliže není, generuje se odstranitelná chyba.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.59 DnumToNum - Převádí dnum na num

RobotWare - OS

Pokračování

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

Kód chyby	Popis
ERR_ARGVALERR	Hodnota je nad 8388608 nebo pod -8388607 nebo není celým číslem (jestliže se používá volitelný argument Integer)
ERR_NUM_LIMIT	Hodnota je nad 3.40282347E+38 nebo pod -3.40282347E+38
ERR_INT_NOTVAL	Hodnota není celé číslo

Syntaxe

```
DnumToNum '('
  [ Value ':=' ] < expression (IN) of dnum >
  ['\' Integer ')'
```

Funkce s vrácenou hodnotou datového typu num.

Související informace

Pro informace o	Viz
Datový typ Dnum	Kontrola informačních štítků
Datový typ Num	Kontrola informačních štítků

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.60 DnumToStr - Převádí numerickou hodnotu na řetězec

Použití

`DnumToStr` (*Numeric To String*) se používá k převodu numerické hodnoty na řetězec.

Základní příklady

Následující příklady názorně ukazují funkci`DnumToStr`.

Příklad 1

```
VAR string str;
str := DnumToStr(0.3852138754655357,3);
```

Proměnné `str` je dána hodnota `"0.385"`.

Příklad 2

```
VAR dnum val;
val:= 0.3852138754655357;
str := DnumToStr(val, 2\Exp);
```

Proměnné `str` je dána hodnota `"3.85E-01"`.

Příklad 3

```
VAR dnum val;
val := 0.3852138754655357;
str := DnumToStr(val, 15);
```

Proměnné `str` je dána hodnota `"0.385213875465536"`.

Příklad 4

```
VAR dnum val;
val:=4294967295.385215;
str := DnumToStr(val, 4);
```

Proměnné `str` je dána hodnota `"4294967295.3852"`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `str`

Numerická hodnota převedená na řetězec s určeným počtem desetinných čísel, s exponentem, jestliže je to požadováno. Numerická hodnota je podle potřeby zaokrouhlena. Desetinná tečka (čárka) je potlačena, jestliže nejsou obsažena žádná desetinná místa.

Argumenty

```
DnumToStr (Val Dec [\Exp])
```

`Val`

Value

Datový typ: `dnum`

Numerická hodnota, která bude převedena.

`Dec`

Decimals

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.60 DnumToStr - Převádí numerickou hodnotu na řetězec

RobotWare - OS

Pokračování

Datový typ: `num`

Počet desetinných míst. Počet desetinných míst nesmí být záporný nebo větší než dostupná řádová přesnost pro numerické hodnoty.

Max počet desetinných míst, která mohou být použita, je 15.

`[\Exp]`

Exponent

Datový typ: `switch`

Použití exponentu ve vrácené hodnotě.

Syntaxe

```
DnumToStr '('
  [ Val ':=' ] <expression (IN) of dnum>
  [ Dec ':=' ] <expression (IN) of num>
  ['\' Exp ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Řetězcové funkce*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Základní prvky*
Převést numerickou hodnotu num na řetězec	*NumToStr - Převádí numerickou hodnotu na řetězec na str 1237*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.61 DotProd - Skalární součin dvou pos vektorů

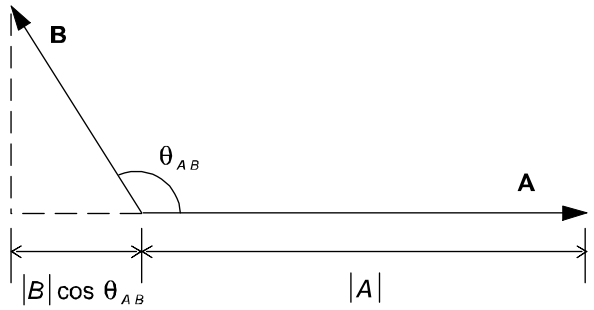
Použití

`DotProd` (*Dot Product*) se používá pro výpočet dot (nebo skalárního) produktu dvou pos vektorů. Typické využití je pro výpočet projekce jednoho vektoru nad jiným nebo pro výpočet úhlu mezi dvěma vektory.

Základní příklady

Následující příklad názorně ukazuje funkci`DotProd`.

Příklad 1



xx0500002449

Dot nebo skalární produkt dvou vektorů **A** a **B** je skalár, který je rovný produktům magnitud **A** a **B** a kosinu úhlu mezi nimi.

$$A \cdot B = |A||B| \cos \theta_{AB}$$

Dot produkt:

- je menší nebo rovný produktu jejich magnitud.
- může být buď kladná nebo záporná kvantita, podle toho, jestli úhel mezi nimi je menší nebo větší než 90 stupňů.
- je rovný produktu magnitudy jednoho vektoru a projekce jiného vektoru nad prvním.
- je nula, když vektory jsou vůči sobě kolmé.

Vektory jsou popsány datovým typem `pos` a dot produkt datovým typem `num`:

```
VAR num dotprod;
VAR pos vector1;
VAR pos vector2;
...
...
vector1 := [1,1,1];
vector2 := [1,2,3];
dotprod := DotProd(vector1, vector2);
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Hodnota dot produktu dvou vektorů.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.61 DotProd - Skalární součin dvou pos vektorů

RobotWare - OS

Pokračování

Argumenty

```
DotProd (Vector1 Vector2)
```

`Vector1`

Datový typ: `pos`

První vektor popsaný datovým typem `pos`.

`Vector2`

Datový typ: `pos`

Druhý vektor popsaný datovým typem `pos`.

Syntaxe

```
DotProd '('
  [Vector1 ':='] <expression (IN) of pos>','
  [Vector2 ':='] <expression (IN) of pos>
  ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Matematika*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.62 DOutput - Čte hodnotu digitálního výstupního signálu

Použití

`DOutput` se používá pro čtení aktuální hodnoty digitálního výstupního signálu.

> **POZNÁMKA**
>
> Všimněte si, že funkce `DOutput` je zděděná funkce, kterou již není nutné používat. Viz příklady alternativních a doporučených způsobů programování.

Základní příklady

Následující příklad názorně ukazuje funkci`DOutput`.

Viz také *Další příklady na str 1146*.

Příklad 1

```
IF DOutput(do2) = 1 THEN ...
...
IF do2 = 1 THEN ...
```

Jestliže aktuální hodnota signálu `do2` je rovna `1`, potom `...`

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dionum`

Aktuální hodnota signálu (0 nebo 1).

Argumenty

```
DOutput (Signal)
```

`Signal`

Datový typ: `signaldo`

Jméno signálu, který bude přečten.

Vykonávání programu

Čtená hodnota signálu závisí na konfiguraci signálu. Jestliže je signál invertován v systémových parametrech, vrácená hodnota touto funkcí je opačná k true hodnotě fyzického kanálu.

Řešení chyb

Následující odstranitelné chyby mohou být vyvolány. Chyby je možné řešit v obslužném programu ERROR. Systémová proměnná `ERRNO` bude nastavena na:

Název	Příčina chyby
`ERR_NO_ALIASIO_DEF`	Proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.
`ERR_NORUNUNIT`	Jestliže není žádný kontakt s jednotkou I/O
`ERR_SIG_NOT_VALID`	Není možný přístup k I/O signálu. Důvodem může být, že I/O jednotka neběží nebo je chyba v konfiguraci (platné pouze pro aplikační sběrnici ICI).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.62 DOutput - Čte hodnotu digitálního výstupního signálu

RobotWare - OS

Pokračování

Další příklady

Více příkladů funkce `DOutput` je názorně uvedeno dole.

Příklad 1

```
IF DOutput(auto_on) <> active THEN ...
...
IF auto_on <> active THEN ...
```

Jestliže aktuální hodnota systémového signálu `auto_on` je `not active` potom `...`, tj. jestliže robot je v ručním provozním režimu, potom `...`

> **POZNÁMKA**
>
> Signál musí být nejprve definován jako výstup systému v systémových parametrech.

Syntaxe

```
DOutput '('
  [ Signal ':=' ] < variable (VAR) of signaldo > ')'
```

Funkce s vrácenou hodnotou datového typu `dionum`.

Související informace

Pro informace o	Viz
Nastavit digitální výstupní signál	*SetDO - Mění hodnotu digitálního výstupního signálu na str 629*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.63 EGMGetState - Získává aktuální stav EGM

Použití

`EGMGetState` získává stav procesu EGM (`EGMid`).

Základní příklady

```
VAR egmident egmID1;
VAR egmstate egmState1:= EGM_STATE_DISCONNECTED;

EGMGetId egmID1;
egmState1 := EGMGetState(egmID1);
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `egmstate`

Aktuální stav procesu EGM, identifikovaného identitou EGM určenou v argumentu.

Argumenty

```
EGMGetState (EGMid)
```

`EGMid`

Datový typ: `egmident`

identita EGM.

Omezení

- `EGMGetState` může být použit pouze v pohybových úlohách RAPID.
- Mechanická jednotka musí být robot.

Syntaxe

```
EGMGetState '('
  [EGMid ':='] < variable (VAR) of egmident >')'
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.64 EulerZYX - Získává euler úhly z orient

Použití

`EulerZYX` (***Euler ZYX rotations***) se používá k získání komponentu Euler úhlu z proměnné typu `orient`.

Základní příklady

Následující příklad názorně ukazuje funkci`EulerZYX`.

Příklad 1

```
VAR num anglex;
VAR num angley;
VAR num anglez;
VAR pose object;
...
...
anglex := EulerZYX(\X, object.rot);
angley := EulerZYX(\Y, object.rot);
anglez := EulerZYX(\Z, object.rot);
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Odpovídající Euler úhel vyjádřený ve stupních, rozsah [-180, 180].

Argumenty

```
EulerZYX ([\X] | [\Y] | [\Z] Rotation)
```

`[\X]`

Datový typ: `switch`

Získává rotaci kolem osy X.

`[\Y]`

Datový typ: `switch`

Získává rotaci kolem osy Y.

`[\Z]`

Datový typ: `switch`

Získává rotaci kolem osy Z.

Upozornění!

Argumenty `\X`, `\Y`, a `\Z` jsou neslučitelné. Jestli není určen žádný z nich, je generována chyba za běhu.

`Rotation`

Datový typ: `orient`

Rotace ve své reprezentaci kvaternionu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
EulerZYX '('
  ['\' X ','] | ['\' Y ','] | ['\' Z ',']
  [Rotation ':='] <expression (IN) of orient> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Matematika*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.65 EventType - Získat typ aktuální události uvnitř jakékoliv událostní rutiny

Použití

`EventType` se může používat v jakékoliv událostní rutině a potom k vrácení aktuálně vykonávaného typu události.

Jestliže `EventType` je volán z jakékoliv rutiny programové úlohy, potom `EventType` vždy vrací 0, což znamená EVENT_NONE.

Základní příklady

Následující příklad názorně ukazuje funkci`EventType`.

Příklad 1

```
TEST EventType()
CASE EVENT_NONE:
  ! Not executing any event
CASE EVENT_POWERON:
  ! Executing POWER ON event
CASE EVENT_START:
  ! Executing START event
CASE EVENT_STOP:
  ! Executing STOP event
CASE EVENT_QSTOP:
  ! Executing QSTOP event
CASE EVENT_RESTART:
  ! Executing RESTART event
CASE EVENT_RESET:
  ! Executing RESET event
CASE EVENT_STEP:
  ! Executing STEP event
ENDTEST
```

Použijte funkci `EventType` uvnitř jakékoliv událostní rutiny, abyste mohli zjistit, která systémová událost, pokud nějaká, je právě vykonávána.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `event_type`

Aktuálně vykonávaný typ události 1 ... 7 nebo 0, jestliže není vykonávána žádná událostní rutina.

Předdefinovaná data

Následující předdefinované symbolické konstanty typu `event_type` se mohou používat pro kontrolu vrácené hodnoty:

```
CONST event_type EVENT_NONE := 0;
CONST event_type EVENT_POWERON := 1;
CONST event_type EVENT_START := 2;
CONST event_type EVENT_STOP := 3;
CONST event_type EVENT_QSTOP:= 4;
CONST event_type EVENT_RESTART := 5;
CONST event_type EVENT_RESET := 6;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
CONST event_type EVENT_STEP := 7;
```

Syntaxe

```
EventType '(' ')'
```

Funkce s vrácenou hodnotou datového typu `event_type`.

Související informace

Pro informace o	Viz
Událostní rutiny všeobecně	*Technická referenční příručka - Systémové parametry*, sekce *Controller - Event Routine*.
Datový typ event_type, předdefinované konstanty	*event_type - Typ událostní rutiny na str 1504*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.66 ExecHandler - Získat typ prováděcího handleru

Použití

`ExecHandler` se může používat ke zjištění, jestli aktuální `RAPID` kód je vykonán v některém handleru rutiny programu `RAPID`.

Základní příklady

Následující příklad názorně ukazuje funkci`ExecHandler`.

Příklad 1

```
TEST ExecHandler()
CASE HANDLER_NONE:
  ! Not executing in any routine handler
CASE HANDLER_BWD:
  ! Executing in routine BACKWARD handler
CASE HANDLER_ERR:
  ! Executing in routine ERROR handler
CASE HANDLER_UNDO:
  ! Executing in routine UNDO handler
ENDTEST
```

Použijte funkci `ExecHandler` ke zjištění, jestli je kód vykonáván v některém typu handleru rutiny nebo nikoliv.

`HANDLER_ERR` bude vrácen, i když je vykonáváno volání v submetodě k chybovému handleru.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `handler_type`

Aktuálně vykonávaný handler typu 1 ... 3 nebo 0, jestliže není vykonáván žádný handler rutiny.

Předdefinovaná data

Následující předdefinované symbolické konstanty typu `handler_type` se mohou používat pro kontrolu vrácené hodnoty:

```
CONST handler_type HANDLER_NONE := 0;
CONST handler_type HANDLER_BWD := 1;
CONST handler_type HANDLER_ERR := 2;
CONST handler_type HANDLER_UNDO := 3;
```

Syntaxe

```
ExecHandler '(' ')'
```

Funkce s vrácenou hodnotou datového typu `handler_type`.

Související informace

Pro informace o	Viz
Typ prováděcího handleru	*handler_type - Typ prováděcího handleru na str 1511*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.67 ExecLevel - Získat prováděcí úroveň

Použití

`ExecLevel` se může použít ke zjištění aktuální prováděcí úrovně pro kód RAPID, který je aktuálně vykonáván.

Základní příklady

Následující příklad názorně ukazuje funkci`ExecLevel`.

Příklad 1

```
TEST ExecLevel()
CASE LEVEL_NORMAL:
  ! Execute on base level
CASE LEVEL_TRAP:
  ! Execute in TRAP routine
CASE LEVEL_SERVICE:
  ! Execute in service, event or system input interrupt routine
ENDTEST
```

Použití funkce `ExecLevel` ke zjištění aktuální prováděcí úrovně.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `exec_level`

Aktuální prováděcí úroveň 0 ... 2.

Předdefinovaná data

Následující předdefinované symbolické konstanty typu `event_level` se mohou používat pro kontrolu vrácené hodnoty:

```
CONST exec_level LEVEL_NORMAL := 0;
CONST exec_level LEVEL_TRAP := 1;
CONST exec_level LEVEL_SERVICE := 2;
```

Syntaxe

```
ExecLevel '(' ')'
```

Funkce s vrácenou hodnotou datového typu `exec_level`.

Související informace

Pro informace o	Viz
Datový typ pro prováděcí úroveň	*exec_level - Úroveň vykonávání na str 1505*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.68 Exp - Vypočítává exponenciální hodnotu

Použití

`Exp` (*Exponential*) se používá pro výpočet exponenciální hodnoty, e^x.

Základní příklady

Následující příklad názorně ukazuje funkci`Exp`.

Příklad 1

```
VAR num x;
VAR num value;
...
value:= Exp( x);
```

hodnota získá exponenciální hodnotu `x`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Exponenciální hodnota e^x.

Argumenty

```
Exp (Exponent)
```

`Exponent`

Datový typ: `num`

Hodnota argumentu exponentu.

Syntaxe

```
Exp '('
  [Exponent ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Matematika*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.69 FileSize - Vyhledat velikost souboru

Použití

`FileSize` se používá k vyhledání velikosti určeného souboru.

Základní příklady

Následující příklad názorně ukazuje funkci`FileSize`.

Viz také *Další příklady na str 1155*.

Příklad 1

```
PROC listfile(string filename)
  VAR num size;
  size := FileSize(filename);
  TPWrite filename+" size: "+NumToStr(size,0)+" Bytes";
ENDPROC
```

Tato procedura vytiskne jméno určeného souboru společně s velikostní specifikací.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Velikost v bajtech.

Argumenty

```
FileSize (Path)
```

`Path`

Datový typ: `string`

Jméno určeného souboru s plnou nebo relativní cestou.

Vykonávání programu

Tato funkce vrací číselný údaj, který uvádí velikost určeného souboru v bajtech.

Je také možné získat stejnou informaci o adresáři.

Další příklady

Základní příklad funkce je názorně uveden dole.

Příklad 1

Tento příklad uvádí všechny soubory větší než `1` kB pod strukturou adresáře "`HOME`:“ včetně všech podadresářů.

```
PROC searchdir(string dirname, string actionproc)
  VAR dir directory;
  VAR string filename;
  IF IsFile(dirname \Directory) THEN
    OpenDir directory, dirname;
    WHILE ReadDir(directory, filename) DO
        ! .. and . is the parent and resp. this directory
        IF filename <> ".." AND filename <> "." THEN
          searchdir dirname+"/"+filename, actionproc;
        ENDIF
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.69 FileSize - Vyhledat velikost souboru

RobotWare - OS

Pokračování

```
    ENDWHILE
    CloseDir directory;
  ELSE
    %actionproc% dirname;
  ENDIF
ERROR
  RAISE;
ENDPROC

PROC listfile(string filename)
  IF FileSize(filename) > 1024 THEN
      TPWrite filename;
  ENDIF
ENDPROC

PROC main()
  ! Execute the listfile routine for all files found under the
  ! tree of HOME:
  searchdir "HOME:","listfile";
ENDPROC
```

Tento program přenáší strukturu adresáře pod "HOME:“ a u každého nalezeného souboru volá proceduru `listfile`. `searchdir` je generická část, která neví nic o začátku hledání nebo kterou rutinu by bylo třeba volat u každého souboru. Používá `IsFile` ke kontrole, jestli našel podadresář nebo soubor a používá mechanismus opožděné vazby pro volání procedury určené v `actionproc` pro všechny nalezené soubory. `actionproc` rutina `listfile` kontroluje, jestli je soubor větší než 1kB.

Řešení chyb

Jestliže soubor neexistuje, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
FileSize '('
  [ Path ':=' ] < expression (IN) of string> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*
Kopírovat soubor	*CopyFile - Kopírovat soubor na str 135*
Zkontrolovat typ souboru	*IsFile - Zkontrolovat typ souboru na str 1207*
Zkontrolujte velikost souborového systému	*FSSize - Vyhledat velikost souborového systému na str 1161*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.70 FileTimeDnum - Získat časovou informaci o souboru

Použití

`FileTimeDnum` se používá k vyhledání poslední modifikace, přístupu nebo změny statutu souboru. Čas je měřen v sekundách od 00:00:00 GMT, 1. ledna 1970. Čas je vrácen jako `dnum` a volitelně také v `stringdig`.

Základní příklad

Následující příklad názorně ukazuje funkci`FileTimeDnum`.

Viz také *Další příklady na str 1159*.

Příklad 1

```
IF FileTimeDnum ("HOME:/mymod.mod" \ModifyTime) > ModTimeDnum
     ("mymod") THEN
  UnLoad "HOME:/mymod.mod";
  Load \Dynamic, "HOME:/mymod.mod";
ENDIF
```

Tento program znovu načítá modul, jestliže zdrojový soubor je novější. Používá `ModTimeDnum` k vyhledání poslední modifikace určeného modulu a porovnává ji s `FileTimeDnum ("HOME:/mymod.mod" \ModifyTime)` u zdroje. Potom, jestliže zdroj je novější, program stáhne a znovu načte modul.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Čas měřený v sekundách od 00:00:00 GMT, 1. ledna 1970.

Argumenty

```
FileTimeDnum ( Path [\ModifyTime] | [\AccessTime] | [\StatCTime]
     [\StrDig])
```

`Path`

Datový typ: `string`

Určený soubor s plnou nebo relativní cestou.

`[\ModifyTime]`

Datový typ: `switch`

Čas poslední modifikace.

`[\AccessTime]`

Datový typ: `switch`

Čas posledního přístupu (čtení, vykonávání nebo modifikace).

`[\StatCTime]`

Datový typ: `switch`

Poslední čas změny statusu souboru (kvalifikace přístupu).

`[\StrDig]`

String Digit

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Datový typ: `stringdig`

Získání času souboru v reprezentaci `stringdig`.

Vykonávání programu

Tato funkce vrací číselný údaj, který uvádí čas od poslední:

- **Modifikace**
- **Přístup**
- **Změna statusu souboru**

určeného souboru.

Je také možné získat stejnou informaci o adresáři.

Další příklady

Více příkladů funkce `FileTimeDnum` **je názorně uvedeno dole.**

Toto je kompletní příklad, který implementuje alarmovou službu pro max 10 souborů.

```
LOCAL RECORD falert
  string filename;
  dnum ftime;
ENDRECORD

LOCAL VAR falert myfiles[10];
LOCAL VAR num currentpos:=0;
LOCAL VAR intnum timeint;

PROC alertInit(num freq)
  currentpos:=0;
  CONNECT timeint WITH mytrap;
  ITimer freq,timeint;
ENDPROC

LOCAL TRAP mytrap
  VAR num pos:=1;
  WHILE pos <= currentpos DO
    IF FileTimeDnum(myfiles{pos}.filename \ModifyTime) >
         myfiles{pos}.ftime THEN
      TPWrite "The file "+myfiles{pos}.filename+" is changed";
    ENDIF
    pos := pos+1;
  ENDWHILE
ENDTRAP

PROC alertNew(string filename)
  currentpos := currentpos+1;
  IF currentpos <= 10 THEN
    myfiles{currentpos}.filename := filename;
    myfiles{currentpos}.ftime := FileTimeDnum (filename
         \ModifyTime);
  ENDIF
ENDPROC
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.70 FileTimeDnum - Získat časovou informaci o souboru

Pokračování

```
PROC alertFree()
  IDelete timeint;
ENDPROC
```

Řešení chyb

Jestliže soubor neexistuje, potom je systémová proměnná ERRNO nastavena na ERR_FILEACC. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
FileTimeDnum '('
[ Path ':=' ] < expression (IN) of string>
[ '\' ModifyTime] |
[ '\' AccessTime] |
[ '\' StatCTime]
[ '\' StrDig ':=' < variable (VAR) of stringdig> ] ')'
```

Funkce s vrácenou hodnotou datového typu dnum.

Související informace

Pro informace o	Viz
Čas poslední modifikace načteného modulu	*ModTimeDnum - Získat čas modifikace souboru pro načtený modul na str 1227*
Řetězec pouze s číslicemi	*stringdig - Řetězec pouze s číslicemi na str 1598*
Porovnat dva řetězce pouze s číslicemi	*StrDigCmp - Porovnat dva řetězce pouze s číslicemi na str 1336*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.71 FSSize - Vyhledat velikost souborového systému

Použití

`FSSize` (*File System Size*) se používá pro vyhledání velikosti souborového systému, ve kterém je umístěn určený soubor. Velikosti v bajtech, kilobajtech nebo megabajtech jsou vraceny jako `num`.

Základní příklad

Následující příklad názorně ukazuje funkci`FSSize`.

Viz také *Další příklady na str 1162*.

Příklad 1

```
PROC main()
  VAR num totalfsyssize;
  VAR num freefsyssize;
  freefsyssize := FSSize("HOME:/spy.log" \Free);
  totalfsyssize := FSSize("HOME:/spy.log" \Total);
  TPWrite NumToStr(((totalfsyssize -
       freefsyssize)/totalfsyssize)*100,0) +" percent used";
ENDPROC
```

Tato procedura vytiskne velikost diskového prostoru používanou na `HOME`: souborový systém (flash disk `/hd0a/`) jako procentuální část.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Velikost v bajtech.

Argumenty

```
FSSize (Name [\Total] | [\Free] [\Kbyte] [\Mbyte])
```

`Name`

Datový typ: `string`

Jméno souboru v souborovém systému, uvedené s plnou nebo relativní cestou.

`[ \Total ]`

Datový typ: `switch`

Vyhledá celkovou velikost prostoru v souborovém systému.

`[ \Free ]`

Datový typ: `switch`

Vyhledá celkovou velikost volného prostoru v souborovém systému.

`[ \Kbyte ]`

Datový typ: `switch`

Převést počet přečtených bajtů na kilobajty, například, velikost dělit 1024.

`[ \Mbyte ]`

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.71 FSSize - Vyhledat velikost souborového systému

RobotWare - OS

Pokračování

Převést počet přečtených bajtů na megabajty, například, velikost dělit 1048576 (1024*1024).

Vykonávání programu

Tato funkce vrací číselný údaj, který uvádí velikost souborového systému, ve kterém je umístěn určený soubor.

Další příklady

Více příkladů funkce FSSize je názorně uvedeno dole.

Příklad 1

```
LOCAL VAR intnum timeint;

LOCAL TRAP mytrap
  IF FSSize("HOME:/spy.log" \Free)/FSSize("HOME:/spy.log" \Total)
       <= 0.1 THEN
    TPWrite "The disk is almost full";
    alertFree;
  ENDIF
ENDTRAP

PROC alertInit(num freq)
  CONNECT timeint WITH mytrap;
  ITimer freq,timeint;
ENDPROC

PROC alertFree()
  IDelete timeint;
ENDPROC
```

Toto je kompletní příklad pro implementaci alarmové služby, která tiskne varování na FlexPendant, když zbývající volný prostor v souborovém systému "HOME:" je menší než 10 %.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_FILEACC	Souborový systém neexistuje
ERR_FILESIZE	Velikost přesahuje max hodnotu celého čísla pro num, 8388608

Syntaxe

```
FSSize '('
  [ Name ':=' ] < expression (IN) of string>
  [ '\' Total ] | [ '\' Free ]
  [ '\' Kbyte ]
  [ '\' Mbyte ] ')'
```

Funkce s vrácenou hodnotou datového typu num.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*
Kopírovat soubor	*CopyFile - Kopírovat soubor na str 135*
Zkontrolovat typ souboru	*IsFile - Zkontrolovat typ souboru na str 1207*
Zkontrolujte velikost souboru	*FileSize - Vyhledat velikost souboru na str 1155*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.72 GetMaxNumberOfCyclicBool - Získat max počet cyklicky vyhodnocovaných logických podmínek

Použití

`GetMaxNumberOfCyclicBool` se používá pro vyhledání max počtu cyklicky vyhodnocovaných logických podmínek, které mohou být připojeny ve stejnou dobu.

Základní příklady

Následující příklad názorně ukazuje funkci`GetMaxNumberOfCyclicBool`.

Příklad 1

```
VAR num maxno := 0;
maxno := GetMaxNumberOfCyclicBool();
TpWrite "Maximum cyclic bool: " \Num:=maxno;
```

Max počet připojených cyklicky vyhodnocovaných logických podmínek je zobrazen na FlexPendantu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Syntaxe

```
GetMaxNumberOfCyclicBool '(' ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Nastavit cyklicky hodnocenou logickou podmínku	*SetupCyclicBool - Nastavit cyklicky hodnocenou logickou podmínku na str 636*
Odstranit cyklicky hodnocenou logickou podmínku	*RemoveCyclicBool - Odstranit cyklicky hodnocenou logickou podmínku na str 534*
Odstranit všechny cyklicky hodnocené logické podmínky	*RemoveAllCyclicBool - Odstranit všechny cyklicky hodnocené logické podmínky na str 533*
Cyklicky hodnocené logické podmínky, *Cyclic bool*.	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.73 GetMecUnitName - Získat jméno mechanické jednotky

Použití

`GetMecUnitName` se používá k získání jména mechanické jednotky s jednou z instalovaných mechanických jednotek jako argumentem. Tato funkce vrací jméno mechanické jednotky jako `string`.

Základní příklady

Následující příklad názorně ukazuje funkci`GetMecUnitName`.

Příklad 1

```
VAR string mecname;
mecname:= GetMecUnitName(ROB1);
```

Pro `mecname` je přidělena hodnota `"ROB1"` jako `string`. Všechny mechanické jednotky (datový typ `mecunit`) jako je `ROB1` jsou předdefinovány v systému.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Vrácenou hodnotou bude jméno mechanické jednotky jako `string`.

Argumenty

```
GetMecUnitName ( MechUnit )
```

`MechUnit`

Mechanical Unit

Datový typ: `mecunit`

`MechUnit` bere jednu z předdefinovaných mechanických jednotek nalezených v konfiguraci.

Syntaxe

```
GetMecUnitName '('
  [ MechUnit ':=' ] < variable (VAR) of mecunit > ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Mechanická jednotka	*mecunit - Mechanická jednotka na str 1531*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.74 GetModalPayLoadMode - Získat hodnotu ModalPayLoadMode

Použití

`GetModalPayLoadMode` se používá pro získání `ModalPayLoadMode`.

Základní příklady

Následující příklad názorně ukazuje funkci`GetModalPayLoadMode`.

Příklad 1

```
IF GetModalPayloadMode() = 1 THEN
  GripLoad piece1;
  MoveL p1, v1000, fine, gripper;
ELSE
  MoveL p1, v1000, fine, tool2 \TLoad:=gripperpiece1;
ENDIF
```

Přečtěte hodnotu `ModalPayLoadMode` ze systému a podle hodnoty použijte odlišný kód k určení zátěže použité v pohybové instrukci.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Vrácenou hodnotou bude nastavení `ModalPayLoadMode` jako `num`.

Syntaxe

```
GetModalPayloadMode '(' ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Systémový parametr *ModalPayLoadMode* pro aktivaci a deaktivaci užitečné zátěže. (Téma Controller, Typ System Misc, Akční hodnoty, *ModalPayLoadMode*)	*Technická referenční příručka - Systémové parametry*
Používání užitečné zátěže v pohybových instrukcích.	*MoveL - Pohybuje robotem lineárně na str 411*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.75 GetMotorTorque - Čte aktuálního točivý moment motoru

Použití

`GetMotorTorque` se používá pro čtení aktuálního točivého momentu motorů robotu a externích os.

`GetMotorTorque` se primárně používá ke zjištění, jestli servo chapadlo drží náklad nebo nikoliv.

Základní příklady

Následující příklad názorně ukazuje funkci`GetMotorTorque`.

Viz také *Další příklady na str 1168*.

Příklad 1

```
VAR num motor_torque2;
motor_torque2 := GetMotorTorque(2);
```

Aktuální točivý moment motoru druhé osy robotu je uložen do `motor_torque2`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Aktuální točivý moment moment motoru v newtonmetrech (Nm) stanovené osy robotu nebo externích os.

Argumenty

```
GetMotorTorque [\MecUnit ] AxisNo
```

`MecUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky, pro kterou bude přečtena osa. Jestliže tento argument je vynechán, bude přečtena osa připojeného robotu.

`AxisNo`

Datový typ: `num`

Číslo osy, která bude přečtena (1 - 6).

Vykonávání programu

Funkce čte aktuální filtrovaný moment motoru použitý na motorech robotu a externích os.

Hodnota momentu motoru může být vidět také jako testovací signál číslo 2000 při použití *Test Signal Viewer* nebo *Tune Master*.

Omezení

Výsledek `GetMotorTorque` se bude měnit podle tření převodů, teploty motoru atd. Dvě měření ve stejné pozici se mohou lišit. Příklad: teplota převodovky může měnit tření a tedy i výsledek.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.75 GetMotorTorque - Čte aktuálního točivý moment motoru
RobotWare - OS
Pokračování

Omezení popsaná nahoře mohou způsobit, že bude nemožné zjistit velmi malé změny v točivém momentu.

Je pouze možné přečíst aktuální moment u mechanických jednotek, které jsou kontrolovány z aktuální programové úlohy. U nepohybových úloh je možné přečíst moment u mechanických jednotek kontrolovaných připojenou pohybovou úlohou.

Další příklady

Následující příklady názorně ukazují funkci`GetMotorTorque`.

Příklad 1

```
VAR num torque_value;
torque_value := GetMotorTorque(\MecUnit:=STN1, 1);
```

Aktuální točivý moment motoru první osy `STN1` je uložen do `torque_value`.

Příklad 2

```
VAR num pre_grip_torque;
VAR num post_grip_torque;
..
MoveJ p10, v1000, fine, Gripper;
! Read the torque for axis 5 before gripping the piece
pre_grip_torque:=GetMotorTorque(5);
! Grip the piece
grip_piece;
! Read the torque for axis 5 after gripping the piece
post_grip_torque:=GetMotorTorque(5);
! Compare torque for axis 5 before and after gripping the piece
piece_gripped:=check_gripped_piece(pre_grip_torque,
     post_grip_torque);
IF piece_gripped = TRUE THEN
  GripLoad piece1;
ELSE
  TPWrite "Failed to grip the piece";
  Stop;
ENDIF
..
```

Aktuální moment motoru osy 5 robotu je přečten před uchopením kusu. Kus je potom uchopen. Moment je přečten ještě jednou a momenty jsou porovnány kvůli zjištění, jestli je v chapadle skutečně další náklad.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_AXIS_PAR`	Parametr osa v instrukci je nesprávný.

Syntaxe

```
GetMotorTorque '('
  ['\' MecUnit ':=' < variable (VAR) of mecunit> ',']
  [AxisNo ':=' ] < expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Čte aktuální úhly motoru	*ReadMotor - Čte aktuální úhly motoru na str 1286*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.76 GetNextCyclicBool - Získat jméno cyklicky hodnocené logické podmínky

Použití

`GetNextCyclicBool` se používá k vyhledání jména připojené cyklicky hodnocené logické podmínky.

Základní příklady

Následující příklad názorně ukazuje funkci`GetNextCyclicBool`.

Příklad 1

```
VAR num listno := 0;
WHILE GetNextCyclicBool(listno, name) DO
TpWrite "Cyclic bool: "+name;
! listno := listno + 1 is done by GetNextCyclicBool
ENDWHILE
```

Jména všech připojených cyklicky hodnocených logických podmínek v systému budou zobrazena na FlexPendantu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Vrácená hodnota `TRUE`, jestliže byla nalezena cyklicky hodnocená logická podmínka, jinak `FALSE`.

Argumenty

```
GetNextCyclicBool(ListNumber Name)
```

`ListNumber`

Datový typ: `num`

Tímto je určeno, které položky připojených cyklicky hodnocených logických podmínek mají být vyhledány. Tato proměnná je vždy navyšována systémem o jednu, aby byl usnadněn přístup k další podmínce v seznamu. První cyklicky hodnocená logická podmínka v seznamu má index 0.

`Name`

Datový typ: `string`

Jméno cyklicky hodnocené logické podmínky.

Syntaxe

```
GetNextCyclicBool '('
  [ ListNumber ':=' ] < variable (VAR) of num> ','
  [ Name ':=' ] < variable (VAR) of string> ','
  ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Nastavit cyklicky hodnocenou logickou podmínku	*SetupCyclicBool - Nastavit cyklicky hodnocenou logickou podmínku na str 636*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Odstranit cyklicky hodnocenou logickou podmínku	*RemoveCyclicBool - Odstranit cyklicky hodnocenou logickou podmínku na str 534*
Odstranit všechny cyklicky hodnocené logické podmínky	*RemoveAllCyclicBool - Odstranit všechny cyklicky hodnocené logické podmínky na str 533*
Cyklicky hodnocené logické podmínky, *Cyclic bool*.	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.77 GetNextMechUnit - Získat jméno a data pro mechanické jednotky

Použití

`GetNextMechUnit` (*Get Next Mechanical Unit*) se používá k vyhledání jména mechanických jednotek v systému robotu. Vedle jména mechanické jednotky může být vyhledáno několik volitelných vlastností mechanické jednotky.

Základní příklady

Následující příklad názorně ukazuje funkci`GetNextMechUnit`.

Viz také *Další příklady na str 1173*.

Příklad 1

```
VAR num listno := 0;
VAR string name := "";

TPWrite "List of mechanical units:";
WHILE GetNextMechUnit(listno, name) DO
  TPWrite name;
  ! listno := listno + 1 is done by GetNextMechUnit
ENDWHILE
```

Jména všech mechanických jednotek dostupných v systému budou zobrazena na FlexPendantu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže mechanická jednotka byla nalezena, jinak `FALSE`.

Argumenty

```
GetNextMechUnit( ListNumber UnitName [\MecRef] [\TCPRob] [\NoOfAxes]
     [\MecTaskNo] [\MotPlanNo] [\Active] [\DriveModule]
     [\OKToDeact])
```

`ListNumber`

Datový typ: `num`

Tímto je určeno, které položky v systémovém interním seznamu mechanických jednotek mají být vyhledány. Tato proměnná je vždy navyšována systémem o jednu, aby byl usnadněn přístup k další jednotce v seznamu. První mechanická jednotka v seznamu má index 0.

`UnitName`

Datový typ: `string`

Jméno mechanické jednotky.

`[\MecRef]`

Datový typ: `mecunit`

Systémová reference k mechanické jednotce.

`[\TCPRob]`

Datový typ: `bool`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

TRUE, jestliže mechanickou jednotkou je TCP robot, jinak FALSE.

[\NoOfAxes]

Datový typ: num

Počet os mechanické jednotky. Hodnota celého čísla.

[\MecTaskNo]

Datový typ: num

Číslo programové úlohy, která kontroluje mechanickou jednotku. Hodnota celého čísla v rozsahu 1-20. Jestliže není kontrolována žádnou programovou úlohou, je vráceno -1.

Toto aktuální připojení je definováno ovladačem domény v systémových parametrech (v některé aplikaci může být znovu definováno za běhu).

[\MotPlanNo]

Datový typ: num

Číslo plánovače pohybů, který kontroluje mechanickou jednotku. Hodnota celého čísla v rozsahu 1-6. Jestliže není kontrolována žádným plánovačem pohybů, je vráceno -1.

Toto připojení je definováno ovladačem domény v systémových parametrech.

[\Active]

Datový typ: bool

TRUE, jestliže mechanická jednotka je aktivní, jinak FALSE.

[\DriveModule]

Datový typ: num

Pohonný modul číslo 1 - 4 použitý touto mechanickou jednotkou.

[\OKToDeact]

Datový typ: bool

Vrátit TRUE, jestliže je povoleno deaktivovat mechanickou jednotku z programu RAPID.

Další příklady

Více příkladů instrukce GetNextMechUnit je názorně uvedeno dole.

Příklad 1

```
VAR num listno := 4;
VAR string name := "";
VAR bool found := FALSE;

found := GetNextMechUnit (listno, name);
```

Jestliže found je nastaveno na TRUE, jméno mechanické jednotky číslo 4 bude v proměnné name, jinak name obsahuje pouze prázdný řetězec.

Syntaxe

```
GetNextMechUnit '('
  [ ListNumber ':=' ] < variable (VAR) of num> ','
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.77 GetNextMechUnit - Získat jméno a data pro mechanické jednotky

RobotWare - OS

Pokračování

```
[ UnitName ':=' ] < variable (VAR) of string> ','
[ '\' MecRef ':=' < variable (VAR) of mecunit> ]
[ '\' TCPRob ':=' < variable (VAR) of bool> ]
[ '\' NoOfAxes ':=' < variable (VAR) of num> ]
[ '\' MecTaskNo ':=' < variable (VAR) of num> ]
[ '\' MotPlanNo ':=' < variable (VAR) of num> ]
[ '\' Active ':=' < variable (VAR) of bool>]
[ '\' DriveModule ':=' < variable (VAR) of num>]
[ '\' OKToDeact ':=' < variable (VAR) of bool>]
')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Mechanická jednotka	*mecunit - Mechanická jednotka na str 1531*
Aktivace/deaktivace mechanických jednotek	*ActUnit - Aktivuje mechanickou jednotku na str 24* *DeactUnit - Deaktivuje mechanickou jednotku na str 150*
Vlastnosti nehodnotových datových typů	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.78 GetNextSym - Získat další odpovídající symbol

Použití

`GetNextSym` (*Get Next Symbol*) se používá společně s `SetDataSearch` k získání datových objektů ze systému.

Základní příklady

Následující příklad názorně ukazuje funkci`GetNextSym`.

Příklad 1

```
VAR datapos block;
VAR string name;
VAR bool truevar:=TRUE;
...
SetDataSearch "bool" \Object:="my.*" \InMod:="mymod"\LocalSym;
WHILE GetNextSym(name,block) DO
  SetDataVal name\Block:=block,truevar;
ENDWHILE
```

Tato akce nastaví všechny lokální datové objekty `bool`, které začínají s `my` v modulu `mymod` na `TRUE`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže nový objekt má být vyhledán, jméno objektu a jeho přiložený blok jsou potom vráceny v jeho argumentech.

`FALSE`, jestliže žádné další objekty nesouhlasí.

Argumenty

```
GetNextSym (Object Block [\Recursive])
```

`Object`

Datový typ: `string`

Proměnná (`VAR` nebo `PERS`) k uložení jména datového objektu, který bude vyhledán.

`Block`

Datový typ: `datapos`

Přiložený blok k objektu.

`[ \Recursive ]`

Datový typ: `switch`

Tímto bude hledání přinuceno vstoupit do bloku dole, například, jestliže akce hledání začala na úrovni úlohy, bude prohledávat také moduly a rutiny pod úlohou.

Syntaxe

```
GetNextSym '('
  [ Object ':=' ] < variable or persistent (INOUT) of string > ','
  [ Block ':='] <variable (VAR) of datapos>
  ['\' Recursive ] ')'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.78 GetNextSym - Získat další odpovídající symbol

RobotWare - OS

Pokračování

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Definovat sadu symbolů ve vyhledávací akci	*SetDataSearch - Definovat sadu symbolů ve vyhledávací sekvenci na str 622*
Získat hodnotu datového objektu.	*GetDataVal - Získat hodnotu datového objektu na str 224*
Nastavit hodnotu datového objektu	*SetDataVal - Nastavit hodnotu datového objektu na str 626*
Nastavit hodnotu mnoha datových objektů	*SetAllDataVal - Nastavit hodnotu všech datovým objektům v definované sadě na str 618*
Související datový typ `datapos`	*datapos - Přiložený blok pro datový objekt na str 1482*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.79 GetNumberOfCyclicBool - Získat číslo cyklicky hodnocené logické podmínky

Použití

`GetNumberOfCyclicBool` se používá k vyhledání čísla připojené cyklicky hodnocené logické podmínky.

Základní příklady

Následující příklad názorně ukazuje funkci`GetNumberOfCyclicBool`.

Příklad 1

```
VAR num listno := 0;
listno := GetNumberOfCyclicBool();
TpWrite "Connected cyclic bool: " \Num:=listno;
```

Počet připojených cyklicky hodnocených logických podmínek je zobrazen na FlexPendantu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Syntaxe

```
GetNumberOfCyclicBool '(' ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Nastavit cyklicky hodnocenou logickou podmínku	*SetupCyclicBool - Nastavit cyklicky hodnocenou logickou podmínku na str 636*
Odstranit cyklicky hodnocenou logickou podmínku	*RemoveCyclicBool - Odstranit cyklicky hodnocenou logickou podmínku na str 534*
Odstranit všechny cyklicky hodnocené logické podmínky	*RemoveAllCyclicBool - Odstranit všechny cyklicky hodnocené logické podmínky na str 533*
Cyklicky hodnocené logické podmínky, *Cyclic bool*.	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.80 GetServiceInfo - Načíst servisní informace ze systému

Použití

`GetServiceInfo` se používá ke čtení servisní informace ze systému. Tato funkce vrací servisní informaci jako `string`.

Základní příklady

Následující příklad názorně ukazuje funkci`GetServiceInfo`.

Viz také *Další příklady na str 1179*.

Příklad 1

```
VAR string mystring;
VAR num mynum;
IF TaskRunRob() THEN
  mystring:=GetServiceInfo(ROB_ID \DutyTimeCnt);
  IF StrToVal(mystring, mynum) = FALSE THEN
    TPWrite "Conversion failed!";
    Stop;
  ENDIF
ENDIF
```

Jestliže úloha kontroluje robot, použijte předdefinovanou proměnnou `ROB_ID` k přečtení provozního počítadla času. Potom převeďte hodnotu řetězce na numerickou hodnotu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Hodnota servisní informace pro určenou mechanickou jednotku. Přečtěte si víc o vrácených hodnotách v *Argumentech* dole.

Argumenty

```
GetServiceInfo (MechUnit [\DutyTimeCnt])
```

`MechUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky, ze které mají být získány informace.

`[\DutyTimeCnt]`

Duty Time Counter

Datový typ: `switch`

Vrací provozní počítadlo času pro mechanickou jednotku použitou v argumentu `MechUnit`. Řetězec s "0“ je vrácen, jestliže tato možnost je použita ve virtuálním řadiči.

Provozní počítadlo času je hodnota v hodinách, kdy mechanické jednotky byly v režimu zapnutých motorů a brzdy byly uvolněny.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Servisní informace jsou přečteny pro použitý volitelný parametr.

Další příklady

Více příkladů jak používat instrukci `GetServiceInfo` je názorně uvedeno dole.

Příklad 1

```
VAR string mystring;
mystring:=GetServiceInfo(ROB_1 \DutyTimeCnt);
TPWrite "DutyTimeCnt for ROB_1: " + mystring;
mystring:=GetServiceInfo(ROB_2 \DutyTimeCnt);
TPWrite "DutyTimeCnt for ROB_2: " + mystring;
mystring:=GetServiceInfo(INTERCH \DutyTimeCnt);
TPWrite "DutyTimeCnt for INTERCH: " + mystring;
mystring:=GetServiceInfo(STN_1 \DutyTimeCnt);
TPWrite "DutyTimeCnt for STN_1: " + mystring;
mystring:=GetServiceInfo(STN_2 \DutyTimeCnt);
TPWrite "DutyTimeCnt for STN_2: " + mystring;
```

Získat informace o provozním počítadle času pro všechny mechanické jednotky ve vícepohybovém systému, a zapsat hodnoty na FlexPendant.

Syntaxe

```
GetServiceInfo '('
  [MechUnit ':=' ] <variable (VAR) of mecunit> ','
  ['\' DutyTimeCnt] ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Mechanická jednotka	Kontrola informačních štítků

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.81 GetSignalOrigin - Získat informaci o původu I/O signálu

Použití

`GetSignalOrigin` se používá k získání informací o původu I/O signálu.

Základní příklady

Následující příklady názorně ukazují funkci`GetSignalOrigin`:

Příklad 1

```
VAR signalorigin myorig;
VAR string signalname;
...
myorig:=GetSignalOrigin(mysignal, signalname);
IF myorig = SIGORIG_NONE THEN
  TPWrite "Signal cannot be used. AliasIO needed.";
ELSEIF myorig = SIGORIG_CFG THEN
  TPWrite "Signal "+signalname+" is defined in I/O configuration.";
ELSEIF myorig = SIGORIG_ALIAS THEN
  TPWrite "Signal is declared in RAPID.";
  TPWrite "Name according to the I/O configuration: "+signalname;
ENDIF
```

Kód nahoře se může používat k určení původu signálu pojmenovaného `mysignal`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `signalorigin`

`signalorigin`, jak je popsán v tabulce dole.

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
0	`SIGORIG_NONE`	Proměnná I/O signálu je deklarována v RAPIDu a nemá žádnou vazbu alias.
1	`SIGORIG_CFG`	Signál je konfigurován v I/O konfiguraci.
2	`SIGORIG_ALIAS`	Proměnná I/O signálu je deklarována v RAPIDu a má vazbu alias na I/O signál konfigurovaný v I/O konfiguraci.

Argumenty

```
GetSignalOrigin Signal SignalName
```

`Signal`

Datový typ: `signalxx`

Jméno signálu. Musí být datového typu `signaldo`, `signaldi`, `signalgo`, `signalgi`, `signalao` nebo `signalai`.

`SignalName`

Datový typ: `string`

Jméno signálu podle I/O konfigurace nebo prázdný řetězec.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Funkce vrací jeden z následujících předdefinovaných počátků signálu: `SIGORIG_NONE`, `SIGORIG_CFG` nebo `SIGORIG_ALIAS`.

Jestliže `SIGORIG_NONE` je vrácen, `SignalName` se skládá z prázdného řetězce.

Jestliže je vrácen `SIGORIG_CFG` nebo `SIGORIG_ALIAS`, argument `SignalName` obsahuje jméno I/O signálu podle I/O konfigurace.

`GetSignalOrigin` se může používat v generických programech pro kontrolu, jestli signál má vazbu alias a jestli je to vazba na správný fyzický I/O signál.

Syntaxe

```
GetSignalOrigin
  [ Signal':=' ] < variable (VAR) of anytype > ','
  [ SignalName ':=' ] < variable (VAR) of string > ';'
```

Související informace

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*
Definování I/O signálů se jménem aliasu	*AliasIO - Definovat V/V (I/O) signál se jménem aliasu na str 28*
Datový typ `signalorigin`	*signalorigin - Popisuje počátek I/O signálu na str 1580*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.82 GetSysInfo - Získat informace o systému

Použití

`GetSysInfo` se používá ke čtení informací o systému. Dostupné informace obsahují sériové číslo, verzi softwaru, jméno verze softwaru, typ robotu, ID řadiče, IP adresu WAN, jazyk řadiče nebo jméno systému.

Základní příklady

Následující příklad názorně ukazuje funkci`GetSysInfo`.

Příklad 1

```
VAR string serial;
VAR string version;
VAR string versionname;
VAR string rtype;
VAR string cid;
VAR string lanip;
VAR string clang;
VAR string sysname;
serial := GetSysInfo(\SerialNo);
version := GetSysInfo(\SWVersion);
versionname := GetSysInfo(\SWVersionName);
rtype := GetSysInfo(\RobotType);
cid := GetSysInfo(\CtrlId);
lanip := GetSysInfo(\LanIp);
clang := GetSysInfo(\CtrlLang);
sysname := GetSysInfo(\SystemName);
```

Sériové číslo bude uloženo do proměnné `serial`, verze RobotWare bude uložena do proměnné `version`, jméno verze RobotWare bude uloženo do proměnné `versionname`, číslo robotu bude uloženo do proměnné `rtype`, ID číslo řadiče bude uloženo do proměnné `cid`, IP adresa WAN bude uložena do proměnné `lanip`, jazyk řadiče bude uložen do proměnné `clang` a jméno aktuálního aktivního systému bude uloženo do `sysname`.

Příklady vrácených řetězců:

Sériové číslo: `24-12345`

Verze softwaru: `ROBOTWARE_6.03.xxxx`

Jméno verze softwaru: `6.03.00.00`

Typ robotu: `IRB 2400-16/1.5 Type A`

ID řadiče: `MyRobot`

IP adresa WAN: `192.168.8.103`

Jazyk: `en`

Jméno systému: `MySystem`

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jeden z: sériové číslo, verze softwaru, jméno verze softwaru, typ robotu, ID řadiče, IP adresa WAN, jazyk řadiče nebo jméno systému. Přečtěte si víc o vrácených hodnotách v *Argumenty na str 1183* dole.

Argumenty

```
GetSysInfo ([\SerialNo] | [\SWVersion] | [\SWVersionName] |
     [\RobotType] | [\CtrlId] | [\LanIp] | [\CtrlLang] |
     [\SystemName])
```

Musí být přítomen jeden z argumentů `SerialNo`, `SWVersion`, `SWVersionName`, `RobotType` , `CtrlId`, `LanIp` `CtrlLang` nebo `SystemName`.

`[\SerialNo]`

Serial Number

Datový typ: `switch`

Vrací sériové číslo.

`[\SWVersion]`

Software Version

Datový typ: `switch`

Vrací media verzi RobotWare, jak je instalována ve složce *PRODUCTS*.

`[\SWVersionName]`

Software Version Name

Datový typ: `switch`

Vrací jméno na displeji media verze RobotWare.

`[\RobotType]`

Datový typ: `switch`

Vrací typ robotu v aktuální nebo připojené úloze. Jestliže mechanická jednotka není TCP robot, je vráceno "-".

`[\CtrlId]`

Controller ID

Datový typ: `switch`

Vrací ID řadiče. Vrací prázdný řetězec, jestliže není určeno ID řadiče. Řetězec s "`VC`" je vrácen, jestliže tato možnost je použita ve virtuálním řadiči.

`[\LanIp]`

Lan Ip address

Datový typ: `switch`

Vrací IP adresu WAN pro řadič. Je vrácen řetězec s "`VC`", jestliže tato možnost je použita ve virtuálním řadiči. Prázdný řetězec je vrácen, jestliže žádná IP adresa WAN není konfigurována v systému.

`[\CtrlLang]`

Controller Language

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.82 GetSysInfo - Získat informace o systému

RobotWare - OS

Pokračování

Vrací jazyk použitý na řadiči.

Vratná hodnota (Vrátit hodnotu)	Jazyk
cs	čeština
zh	čínština (zjednodušená čínština, kontinentální čínština)
da	dánština
nl	holandština
cs	Angličtina
fi	finština
fr	francouzština
de	němčina
hu	maďarština
it	italština
ja	japonština
ko	korejština
pl	polština
pt	portugalština (brazilská portugalština)
ro	rumunština
ru	ruština
sl	Slovinština
es	španělština
sv	švédština
tr	turečtina

`[\SystemName]`

Datový typ: `switch`

Vrací jméno aktivního systému.

Syntaxe

```
GetSysInfo '('
  ['\'SerialNo]
  | ['\' SWVersion]
  | ['\' SWVersionName]
  | ['\' RobotType]
  | ['\' CtrlId]
  | ['\' LanIp]
  | ['\' CtrlLang]
  | ['\' SystemName] ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Otestovat identitu systému	*IsSysId - Otestovat systémovou identitu na str 1220*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.83 GetTaskName - Získává jméno a číslo aktuální úlohy

Použití

`GetTaskName` se používá k získání identity aktuální programové úlohy s jejím jménem a číslem.

Je také možné od některé *nepohybové úlohy* získat jméno a číslo její připojené *pohybové úlohy*. U systému *MultiMove* definuje systémový parametr *Controller/Tasks/Use Mechanical Unit Group* připojenou *pohybovou úlohu* a v systému základny je hlavní úlohou vždy připojená *pohybová úloha* z jakékoliv jiné úlohy.

Základní příklady

Následující příklady názorně ukazují funkci`GetTaskName`.

Příklad 1

```
VAR string taskname;
...
taskname := GetTaskName();
```

Jméno aktuální úlohy je vráceno do proměnné `taskname`.

Příklad 2

```
VAR string taskname;
VAR num taskno;
...
taskname := GetTaskName(\TaskNo:=taskno);
```

Jméno aktuální úlohy je vráceno do proměnné `taskname`. Identita celého čísla úlohy je uložena do proměnné `taskno`.

Příklad 3

```
VAR string taskname;
VAR num taskno;
...
taskname := GetTaskName(\MecTaskNo:=taskno);
```

Jestliže aktuální úloha je typu *nepohybová úloha*, jméno připojené pohybové úlohy je vráceno do proměnné `taskname`. Numerická identita připojené pohybové úlohy je uložena do proměnné `taskno`.

Jestliže aktuální úloha kontroluje některé mechanické jednotky, jméno aktuální úlohy je vráceno do proměnné `taskname`. Numerická identita úlohy je uložena do proměnné `taskno`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Jméno úlohy, ve které je funkce vykonávána nebo jméno připojené pohybové úlohy.

Argumenty

```
GetTaskName ( [\TaskNo] | [\MecTaskNo] )
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.83 GetTaskName - Získává jméno a číslo aktuální úlohy

RobotWare - OS

Pokračování

`[\TaskNo]`

Datový typ: `num`

Vraťte jméno aktuální úlohy (stejná funkčnost, jestliže není použit žádný z přepínačů `\TaskNo` or `\MecTaskNo`). Získejte také identitu aktuální úlohy reprezentované jako celé číslo. Vrácená čísla budou v rozsahu 1-20.

`[\MecTaskNo]`

Datový typ: `num`

Vraťte jméno připojené pohybové úlohy nebo jméno aktuální pohybové úlohy. Získejte také identitu připojené nebo aktuální pohybové úlohy reprezentované jako hodnota celého čísla. Vrácená čísla budou v rozsahu 1-20.

Syntaxe

```
GetTaskName '('
  [ \TaskNo ':=' ] < variable (VAR) of num >
  [ \MecTaskNo ':=' ] < variable (VAR) of num > ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Víceúlohový	*Technická referenční příručka - Přehled RAPID*, sekce*Přehled RAPID - Souhrn RAPID Multitasking* *Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Multitasking*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.84 GetTime - Čte přesný čas jako numerickou hodnotu

Použití

`GetTime` se používá ke čtení určeného komponentu aktuálního systémového času jako numerické hodnoty.

`GetTime` se může použít pro:

- provádění činnosti programem v určité době
- provádění určitých činností v pracovním dni
- vyloučení provádění určitých činností o víkendu
- reagování na chyby odlišně podle denní doby.

Základní příklady

Následující příklad názorně ukazuje funkci`GetTime`.

Viz také *Další příklady na str 1188*.

Příklad 1

```
hour := GetTime(\Hour);
```

Aktuální čas je uložen do proměnné `hour`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Jeden ze čtyř časových komponentů uvedených dole.

Argument

```
GetTime ( [\WDay] | [\Hour] | [\Min] | [\Sec] )
```

[\WDay]

Datový typ: `switch`

Vraťte aktuální den v týdnu. Rozsah: 1 až 7 (pondělí až neděle).

[\Hour]

Datový typ: `switch`

Vraťte aktuální hodinu. Rozsah: 0 až 23.

[\Min]

Datový typ: `switch`

Vraťte aktuální minutu. Rozsah: 0 až 59.

[\Sec]

Datový typ: `switch`

Vraťte aktuální sekundu. Rozsah: 0 až 59.

Jeden z argumentů musí být určen, jinak se vykonávání programu zastaví s chybovou zprávou.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.84 GetTime - Čte přesný čas jako numerickou hodnotu

RobotWare - OS

Pokračování

Další příklady

Více příkladů funkce `GetTime` je názorně uvedeno dole.

Příklad 1

```
weekday := GetTime(\WDay);
hour := GetTime(\Hour);
IF weekday < 6 AND hour >6 AND hour < 16 THEN
  production;
ELSE
  maintenance;
ENDIF
```

Jestliže je pracovní den a čas mezi 7:00 a 15:59, robot provádí výrobu. Ve všech jiných časech je robot v režimu údržby.

Syntaxe

```
GetTime '('
['\' WDay ]
| [ '\' Hour ]
| [ '\' Min ]
| [ '\' Sec ] ')'
```

Funkce s vrácenou hodnotou typu `num`.

Související informace

Pro informace o	Viz
Instrukce k datumu a času	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Systémový čas*
Nastavování systémových hodin	*Návod k použití - IRC5 s jednotkou FlexPendant*, sekce *Změna nastavení FlexPendant*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.85 GInput - Číst hodnotu skupinového vstupního signálu

Použití

`GInput` se používá pro čtení aktuální hodnoty skupiny digitálních vstupních signálů.

> **POZNÁMKA**
>
> Všimněte si, že funkce `GInput` je zděděná funkce, kterou již není nutné používat. Viz příklady alternativních a doporučených způsobů programování.

Základní příklady

Následující příklad názorně ukazuje funkci`GInput`.

Příklad 1

```
IF GInput(gi2) = 5 THEN ...
...
IF gi2 = 5 THEN ...
```

Jestliže aktuální hodnota signálu `gi2` je rovna `5`, potom `...`

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Aktuální hodnota signálu (kladné celé číslo).

Hodnoty každého signálu ve skupině jsou přečteny a interpretovány jako nepodepsané binární číslo. Toto binární číslo je potom převedeno na celé číslo.

Vrácená hodnota leží v rámci rozsahu, který závisí na počtu signálů ve skupině.

Počet signálů	Přípustné hodnoty
1	0-1
2	0-3
3	0-7
4	0-15
5	0-31
6	0-63
7	0-127
8	0-255
9	0-511
10	0-1023
11	0-2047
12	0-4095
13	0-8191
14	0-16383
15	0-32767
16	0-65535
17	0-131071

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.85 GInput - Číst hodnotu skupinového vstupního signálu

Pokračování

Počet signálů	Přípustné hodnoty
18	0-262143
19	0-524287
20	0-1048575
21	0-2097151
22	0-4194303
23	0-8388607

Argumenty

```
GInput (Signal)
```

`Signal`

Datový typ: `signalgi`

Jméno skupiny signálů, které má být přečteno.

Syntaxe

```
GInput '('
  [ Signal ':=' ] < variable (VAR) of signalgi > ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Přečíst hodnotu skupinového vstupního signálu s více než 23 bity	*GInputDnum - Přečíst hodnotu skupinového vstupního signálu na str 1191*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.86 GInputDnum - Přečíst hodnotu skupinového vstupního signálu

Použití

`GInputDnum` se používá pro čtení aktuální hodnoty skupiny digitálních vstupních signálů větších než 23 bitů.

Základní příklady

Následující příklady názorně ukazují funkci`GInputDnum`.

Příklad 1

```
IF GInputDnum(gi2) = 55 THEN ...
```

Jestliže aktuální hodnota signálu `gi2` je rovna `55`, potom `...`

Příklad 2

```
IF GInputDnum(gi2) = 4294967295 THEN ...
```

Jestliže aktuální hodnota signálu `gi2` je rovna `4294967295`, potom `...`

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Aktuální hodnota signálu (kladné celé číslo).

Hodnoty každého signálu ve skupině jsou přečteny a interpretovány jako nepodepsané binární číslo. Toto binární číslo je potom převedeno na celé číslo.

Vrácená hodnota leží v rámci rozsahu, který závisí na počtu signálů ve skupině.

Počet signálů	Přípustné hodnoty
1	0-1
2	0-3
3	0-7
4	0-15
5	0-31
6	0-63
7	0-127
8	0-255
9	0-511
10	0-1023
11	0-2047
12	0-4095
13	0-8191
14	0-16383
15	0-32767
16	0-65535
17	0-131071
18	0-262143

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.86 GInputDnum - Přečíst hodnotu skupinového vstupního signálu

RobotWare - OS

Pokračování

Počet signálů	Přípustné hodnoty
19	0-524287
20	0-1048575
21	0-2097151
22	0-4194303
23	0-8388607
24	0-16777215
25	0-33554431
26	0-67108863
27	0-134217727
28	0-268435455
29	0-536870911
30	0-1073741823
31	0-2147483647
32	0-4294967295

Argumenty

```
GInputDnum (Signal)
```

`Signal`

Datový typ: `signalgi`

Jméno skupiny signálů, které má být přečteno.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
GInputDnum '('
  [ Signal ':=' ] < variable (VAR) of signalgi > ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Přečíst hodnotu skupinového vstupního signálu	*GInput - Číst hodnotu skupinového vstupního signálu na str 1189*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.87 GOutput - Čte hodnotu skupiny digitálních výstupních signálů

Použití

`GOutput` se používá pro čtení aktuální hodnoty skupiny digitálních výstupních signálů.

Základní příklady

Následující příklad názorně ukazuje funkci`GOutput`.

Příklad 1

```
IF GOutput(go2) = 5 THEN ...
```

Jestliže aktuální hodnota signálu `go2` je rovna `5`, potom `...`

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Aktuální hodnota signálu (kladné celé číslo).

Hodnoty každého signálu ve skupině jsou přečteny a interpretovány jako nepodepsané binární číslo. Toto binární číslo je potom převedeno na celé číslo.

Vrácená hodnota leží v rámci rozsahu, který závisí na počtu signálů ve skupině.

Počet signálů	Přípustná hodnota
1	0-1
2	0-3
3	0-7
4	0-15
5	0-31
6	0-63
7	0-127
8	0-255
9	0-511
10	0-1023
11	0-2047
12	0-4095
13	0-8191
14	0-16383
15	0-32767
16	0-65535
17	0-131071
18	0-262143
19	0-524287
20	0-1048575
21	0-2097151

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Počet signálů	Přípustná hodnota
22	0-4194303
23	0-8388607

Argumenty

```
GOutput (Signal)
```

`Signal`

Datový typ: `signalgo`

Jméno skupiny signálů, které má být přečteno.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
GOutput '('
  [ Signal ':=' ] < variable (VAR) of signalgo > ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Nastavit skupinu výstupních signálů	*SetGO - Mění hodnotu skupiny digitálních výstupních signálů na str 631*
Přečíst skupinu výstupních signálů	*GOutputDnum - Přečíst hodnotu skupinového výstupního signálu na str 1196*
Přečíst skupinu vstupních signálů	*GInputDnum - Přečíst hodnotu skupinového vstupního signálu na str 1191*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.88 GOutputDnum - Přečíst hodnotu skupinového výstupního signálu

Použití

`GOutputDnum` se používá pro čtení aktuální hodnoty skupiny digitálních výstupních signálů větších než 23 bitů.

Základní příklady

Následující příklady názorně ukazují funkci`GOutputDnum`.

Příklad 1

```
IF GOutputDnum(go2) = 55 THEN ...
```

Jestliže aktuální hodnota signálu `go2` je rovna `55`, potom `...`

Příklad 2

```
IF GOutputDnum(go2) = 4294967295 THEN ...
```

Jestliže aktuální hodnota signálu `go2` je rovna `4294967295`, potom `...`

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Aktuální hodnota signálu (kladné celé číslo).

Hodnoty každého signálu ve skupině jsou přečteny a interpretovány jako nepodepsané binární číslo. Toto binární číslo je potom převedeno na celé číslo.

Vrácená hodnota leží v rámci rozsahu, který závisí na počtu signálů ve skupině.

Počet signálů	Přípustné hodnoty
1	0-1
2	0-3
3	0-7
4	0-15
5	0-31
6	0-63
7	0-127
8	0-255
9	0-511
10	0-1023
11	0-2047
12	0-4095
13	0-8191
14	0-16383
15	0-32767
16	0-65535
17	0-131071
18	0-262143

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Počet signálů	Přípustné hodnoty
19	0-524287
20	0-1048575
21	0-2097151
22	0-4194303
23	0-8388607
24	0-16777215
25	0-33554431
26	0-67108863
27	0-134217727
28	0-268435455
29	0-536870911
30	0-1073741823
31	0-2147483647
32	0-4294967295

Argumenty

```
GOutputDnum (Signal)
```

`Signal`

Datový typ: `signalgo`

Jméno skupiny signálů, které má být přečteno.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` **bude nastavena na:**

`ERR_NO_ALIASIO_DEF` **jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí** `AliasIO`**.**

`ERR_NORUNUNIT` **jestliže není žádný kontakt s jednotkou I/O.**

`ERR_SIG_NOT_VALID` **jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).**

Syntaxe

```
GOutputDnum '('
  [ Signal ':=' ] < variable (VAR) of signalgo > ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`**.**

Související informace

Pro informace o	Viz
Nastavit skupinu výstupních signálů	*SetGO - Mění hodnotu skupiny digitálních výstupních signálů na str 631*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.88 GOutputDnum - Přečíst hodnotu skupinového výstupního signálu

RobotWare - OS

Pokračování

Pro informace o	Viz
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID,* sekce *Principy pohybu a I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.89 HexToDec - Převést z šestnáctkového na desítkový formát

Použití

`HexToDec` se používá k převodu čísla určeného v čitelném řetězci v základně 16 do základny 16.

Vstupní řetězec by měl být konstruován ze znakové sady [0-9,A-F,a-f].

Tato rutina zpracovává čísla od 0 do 9223372036854775807dec nebo 7FFFFFFFFFFFFFFF hex.

Základní příklady

Následující příklad názorně ukazuje funkci`HexToDec`.

Příklad 1

```
VAR string str;

str := HexToDec("5F5E0FF");
```

Proměnné `str` je dána hodnota "99999999".

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Řetězec převedený do desítkové reprezentace daného čísla v inparametrovém řetězci.

Argumenty

```
HexToDec ( Str )
```

`Str`

String

Datový typ: `string`

Řetězec k převedení.

Syntaxe

```
HexToDec '('
  [ Str ':=' ] <expression (IN) of string> ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Řetězcové funkce*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Základní prvky*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.90 IndInpos - Nezávislá osa v pozičním statutu

Použití

`IndInpos` se používá pro testování, jestli nezávislá psa dosáhla zvolené pozice.

Základní příklady

Následující příklad názorně ukazuje funkci`IndInpos`.

Příklad 1

```
IndAMove Station_A,1\ToAbsNum:=90,20;
WaitUntil IndInpos(Station_A,1) = TRUE;
WaitTime 0.2;
```

Čekejte, až osa `1` od `Station_A` bude v pozici 90 stupňů.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Tabulka popisuje vrácené hodnoty z `IndInpos`:

Vratná hodnota (Vrátit hodnotu)	Statut osy
`TRUE`	V pozici a má nulovou rychlost.
`FALSE`	Není v pozici a/nebo nemá nulovou rychlost.

Argumenty

```
IndInpos ( MecUnit Axis )
```

`MecUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`Axis`

Datový typ: `num`

Číslo aktuální osy pro mechanickou jednotku (1-6).

Omezení

Nezávislá osa vykonávaná s instrukcí `IndCMove` vždy vrací hodnotu `FALSE`, i když rychlost je nastavena na nulu.

Čas čekání 0,2 sek. by měl být přidán po instrukci, aby bylo zajištěno, že bylo dosaženo správného statutu. Tento časový úsek by měl být delší u externích os se špatnou činností.

Řešení chyb

Jestliže osa není aktivována, systémová proměnná `ERRNO` je nastavena na `ERR_AXIS_ACT`.

Jestliže osa není v nezávislém režimu, systémová proměnná `ERRNO` je nastavena na `ERR_AXIS_IND`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Tyto chyby mohou být zpracovány v chybovém handleru.

Syntaxe

```
IndInpos '('
  [ MecUnit ':=' ] < variable (VAR) of mecunit> ','
  [ Axis ':=' ] < expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Nezávislé osy všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce *Principy pohybu a I/O - Polohování během vykonávání programu*
Ostatní nezávislé instrukce a funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Zkontrolovat stav rychlosti u nezávislých os	*IndSpeed - Status nezávislé rychlosti na str 1202*
Definování nezávislých spojů	*Technická referenční příručka - Systémové parametry*, sekce *Pohyb - Rameno*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.91 IndSpeed - Status nezávislé rychlosti

Použití

`IndSpeed` se používá pro testování, jestli nezávislá osa dosáhla zvolené rychlosti.

Základní příklady

Následující příklad názorně ukazuje funkci`IndSpeed`.

Příklad 1

```
IndCMove Station_A, 2, 3.4;
WaitUntil IndSpeed(Station_A,2 \InSpeed) = TRUE;
WaitTime 0.2;
```

Čekejte, až osa `2` od `Station_A` dosáhne rychlosti 3,4 stupně/s.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Tabulka popisuje vrácené hodnoty z `IndSpeed \IndSpeed`:

Vratná hodnota (Vrátit hodnotu)	Statut osy
`TRUE`	Bylo dosaženo zvolené rychlosti.
`FALSE`	Nebylo dosaženo zvolené rychlosti.

Tabulka popisuje vrácené hodnoty z `IndSpeed \ZeroSpeed:`

Vratná hodnota (Vrátit hodnotu)	Statut osy
`TRUE`	Nulová rychlost.
`FALSE`	Nenulová rychlost

Argumenty

```
IndSpeed ( MecUnit Axis [ \InSpeed ] | [ \ZeroSpeed ] )
```

`MecUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

`Axis`

Datový typ: `num`

Číslo aktuální osy pro mechanickou jednotku (1-6).

`[ \InSpeed ]`

Datový typ: `switch`

`IndSpeed` vrací hodnotu `TRUE`, jestliže osa dosáhla zvolené rychlosti, jinak `FALSE`.

`[ \ZeroSpeed ]`

Datový typ: `switch`

`IndSpeed` vrací hodnotu `TRUE`, jestliže osa má nulovou rychlost, jinak `FALSE`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže oba argumenty `\InSpeed` a `\ZeroSpeed` jsou vynechány, zobrazí se chybová zpráva.

Omezení

Funkce `IndSpeed\InSpeed` vždy vrátí hodnotu `FALSE` v následujících situacích:

- Robot je v ručním režimu se sníženou rychlostí.
- Rychlost je snížena pomocí instrukce `VelSet`.
- Rychlost je snížena z okna produkce.

Čas čekání 0,2 sek. by měl být přidán po instrukci, aby bylo zajištěno, že bylo dosaženo správného statutu. Tento časový úsek by měl být delší u externích os se špatnou činností.

Řešení chyb

Jestliže osa není aktivována, systémová proměnná `ERRNO` je nastavena na `ERR_AXIS_ACT`.

Jestliže osa není v nezávislém režimu, systémová proměnná `ERRNO` je nastavena na `ERR_AXIS_IND`.

Tyto chyby mohou být zpracovány v chybovém handleru.

Syntaxe

```
IndSpeed '('
  [ MecUnit ':=' ] < variable (VAR) of mecunit> ','
  [ Axis ':=' ] < expression (IN) of num>
  [ '\' InSpeed ] | [ '\' ZeroSpeed ] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Nezávislé osy všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Polohování během vykonávání programu*
Ostatní nezávislé instrukce a funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Další příklady	*IndCMove - Nezávislý soustavný pohyb na str 257*
Zkontrolovat stav pozice u nezávislých os	*IndInpos - Nezávislá osa v pozičním statutu na str 1200*
Definování nezávislých spojů	*Technická referenční příručka - Systémové parametry*, sekce *Pohyb - Rameno*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.92 IOUnitState - Získat aktuální stav I/O jednotky

Použití

`IOUnitState` se používá ke zjištění aktuálního stavu I/O jednotky. Její fyzický stav a logický stav definují status pro I/O jednotku.

Základní příklady

Následující příklady názorně ukazují funkci`IOUnitState`.

Příklad 1

```
IF (IOUnitState("UNIT1" \Phys)=IOUNIT_PHYS_STATE_RUNNING) THEN
  ! Possible to access some signal on the I/O unit
ELSE
  ! Read/Write some signal on the I/O unit result in error
ENDIF
```

Byl proveden test, jestli I/O jednotka `UNIT1` je aktivní a běží.

Příklad 2

```
IF (IOUnitState("UNIT1" \Logic)=IOUNIT_LOG_STATE_DISABLED) THEN
  ! Unit is disabled by user from RAPID or FlexPendant
ELSE
  ! Unit is enabled.
ENDIF
```

Byl proveden test, jestli I/O jednotka `UNIT1` je vypnuta.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `iounit_state`

Vrácená hodnota má různé hodnoty podle toho, jestli jsou použity volitelné argumenty `\Logic` nebo `\Phys` nebo jestli není použit vůbec žádný volitelný argument.

Logické stavy I/O jednotky popisují stav a uživatel může přikázat I/O jednotku dovnitř. Stav I/O jednotky je definován v tabulce dole při používání volitelného argumentu `\Logic`.

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
10	IOUNIT_LOG_STATE_DISABLED	I/O jednotka je vypnuta uživatelem z RAPIDu, FlexPendantu nebo systémových parametrů.
11	IOUNIT_LOG_STATE_ENABLED	I/O jednotka je zapnuta uživatelem z RAPIDu, FlexPendantu nebo systémových parametrů. Výchozí hodnota po spuštění.

Když je I/O jednotka logicky zapnuta uživatelem a ovladač aplikační sběrnice se snaží vzít I/O jednotku do fyzického stavu `IOUNIT_PHYS_STATE_RUNNING`, I/O jednotka by se mohla dostat do jiných stavů z různých důvodů (viz tabulka dole).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Stav I/O jednotky, jak je definován v tabulce dole, když se používá volitelný argument `\Phys`.

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
20	IOUNIT_PHYS_STATE_DEACTIVATED	I/O jednotka neběží, vypnuta uživatelem
21	IOUNIT_PHYS_STATE_RUNNING	I/O jednotka běží
22	IOUNIT_PHYS_STATE_ERROR	I/O jednotka nepracuje kvůli chybě za běhu
23	IOUNIT_PHYS_STATE_UNCONNECTED	I/O jednotka je konfigurována, ale není připojena k I/O sběrnici nebo I/O sběrnice je zastavena
24	IOUNIT_PHYS_STATE_UNCONFIGURED	I/O jednotka není konfigurována, ale je připojena k I/O sběrnici. 1)
25	IOUNIT_PHYS_STATE_STARTUP	I/O jednotka je ve spouštěcím režimu. 1)
26	IOUNIT_PHYS_STATE_INIT	I/O jednotka byla vytvořena. 1)

POZNÁMKA

Stav I/O jednotky je definován v tabulce dole, když se nepoužívá žádný z volitelných argumentů `\Phys` nebo `\Logic`.

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
1	IOUNIT_RUNNING	I/O jednotka je aktivní a běží
2	IOUNIT_RUNERROR	I/O jednotka nepracuje kvůli chybě za běhu
3	IOUNIT_DISABLE	I/O jednotka je vypnuta uživatelem z RAPIDu nebo FlexPendantu.
4	IOUNIT_OTHERERR	Jiná konfigurace nebo chyby spouštění

1) Není možné získat tento stav v programu RAPID s aktuální verzí Robotware - OS.

Argumenty

```
IOUnitState (UnitName [\Phys] | [\Logic])
```

`UnitName`

Datový typ: `string`

Jméno I/O jednotky, která má být zkontrolováno (se stejným jménem jako konfigurováno).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.92 IOUnitState - Získat aktuální stav I/O jednotky

RobotWare - OS

Pokračování

`[\Phys]`

Fyzický

Datový typ: `switch`

Jestliže se používá tento parametr, je přečten fyzický stav I/O jednotky.

`[\Logic]`

Logický

Datový typ: `switch`

Jestliže se používá tento parametr, je přečten logický stav I/O jednotky.

Syntaxe

```
IOUnitState '('
  [ UnitName ':=' ] < expression (IN) of string >
  [ '\' Phys] | [ '\' Logic] ')'
```

Funkce s vrácenou hodnotou datového typu `iounit_state`.

Související informace

Pro informace o	Viz
Stav I/O jednotky	*iounit_state - Stav I/O jednotky na str 1519*
Zapnout I/O jednotku	*IOEnable - Aktivovat I/O jednotku na str 281*
Vypnout I/O jednotku	*IODisable - Deaktivovat I/O jednotku na str 278*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.93 IsFile - Zkontrolovat typ souboru

Použití

Funkce `IsFile` získává informace o jmenovaném souboru nebo adresáři a kontroluje, jestli je stejný jako určený typ. Jestliže není určen žádný typ, je provedena pouze kontrola existence.

Argument cesty určuje soubor. Oprávnění ke čtení, zápisu nebo provedení jmenovaného souboru se nevyžaduje, ale všechny adresáře uvedené ve jméně cesty vedoucí k souboru musí být volné k prohledávání.

Základní příklady

Následující příklad názorně ukazuje funkci`IsFile`.

Viz také *Další příklady na str 1208*.

Příklad 1

```
PROC printFT(string filename)
  IF IsFile(filename \Directory) THEN
    TPWrite filename+" is a directory";
    RETURN;
  ENDIF
  IF IsFile(filename \Fifo) THEN
    TPWrite filename+" is a fifo file";
    RETURN;
  ENDIF
  IF IsFile(filename \RegFile) THEN
    TPWrite filename+" is a regular file";
    RETURN;
  ENDIF
  IF IsFile(filename \BlockSpec) THEN
    TPWrite filename+" is a block special file";
    RETURN;
  ENDIF
  IF IsFile(filename \CharSpec) THEN
    TPWrite filename+" is a character special file";
    RETURN;
  ENDIF
ENDPROC
```

Tento příklad vytiskne `filename` a typ určeného souboru na FlexPendantu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Funkce vrátí `TRUE`, jestliže určený typ a aktuální typ se shodují, jinak `FALSE`. Jestliže není určen žádný typ, vrací `TRUE`, jestliže soubor existuje, a jinak `FALSE`.

Argumenty

```
IsFile (Path [\Directory] [\Fifo] [\RegFile] [\BlockSpec]
     [\CharSpec])
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.93 IsFile - Zkontrolovat typ souboru

RobotWare - OS

Pokračování

`Path`

Datový typ: `string`

Určený soubor s plnou nebo relativní cestou.

`[ \Directory ]`

Datový typ: `switch`

Je soubor adresářem.

`[ \Fifo ]`

Datový typ: `switch`

Je soubor souborem fifo.

`[ \RegFile ]`

Datový typ: `switch`

Je soubor regulérním souborem, tj. normálním nebo ASCII souborem.

`[ \BlockSpec ]`

Datový typ: `switch`

Je soubor speciálním souborem bloku.

`[ \CharSpec ]`

Datový typ: `switch`

Je soubor speciálním souborem znaků.

Vykonávání programu

Tato funkce vrací `bool`, který určuje shodu nebo nikoliv.

Další příklady

Více příkladů funkce `IsFile` je názorně uvedeno dole.

Příklad 1

Tento příklad implementuje generický přechod funkce struktury adresáře.

```
PROC searchdir(string dirname, string actionproc)
  VAR dir directory;
  VAR string filename;
  IF IsFile(dirname \Directory) THEN
    OpenDir directory, dirname;
    WHILE ReadDir(directory, filename) DO
        ! .. and . is the parent and resp. this directory
        IF filename <> ".." AND filename <> "." THEN
          searchdir dirname+"/"+filename, actionproc;
        ENDIF
    ENDWHILE
    CloseDir directory;
  ELSE
    %actionproc% dirname;
  ENDIF
ERROR
  RAISE;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
ENDPROC

PROC listfile(string filename)
  TPWrite filename;
ENDPROC

PROC main()
  ! Execute the listfile routine for all files found under the
  ! tree of HOME:
  searchdir "HOME:","listfile";
ENDPROC
```

Tento program přenáší strukturu adresáře pod "HOME:“ a u každého nalezeného souboru volá proceduru `listfile`. `searchdir` je generická část, která neví nic o začátku hledání nebo kterou rutinu by bylo třeba volat u každého souboru. Používá `IsFile` ke kontrole, jestli našel podadresář nebo soubor a používá mechanismus opožděné vazby pro volání procedury určené v `actionproc` pro všechny nalezené soubory. Rutina `actionproc` by měla být procedurou s jedním parametrem typu `string`.

Řešení chyb

Jestliže soubor neexistuje a existuje určený typ, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`. Tato chyba může být potom ošetřena v chybovém handleru.

Omezení

Tuto funkci není možné použít proti sériovým kanálům nebo aplikačním sběrnicím.

Jestliže se použije proti `FTP` nebo `NFS` diskům, existence souboru nebo informace o typu nejsou vždy aktualizovány. Aby byly získány správné informace, může být žádoucí explicitní příkaz proti prohledávané cestě (s instrukcí `Open`) před použitím `IsFile`.

Syntaxe

```
Isfile '('
  [ Path ':=' ] < expression (IN) of string>
  [ '\' Directory ]
  | [ '\' Fifo ]
  | [ '\' RegFile ]
  | [ '\' BlockSpec ]
  | [ '\' CharSpec ] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Adresář	*dir - Struktura adresáře souborů na str 1484*
Otevřít adresář	*OpenDir - Otevřít adresář na str 444*
Zavřít adresář	*CloseDir - Zavřít adresář na str 125*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.93 IsFile - Zkontrolovat typ souboru

RobotWare - OS

Pokračování

Pro informace o	Viz
Načíst adresář	*ReadDir - Přečíst další vstupní data v adresáři na str 1283*
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*
Kopírovat soubor	*CopyFile - Kopírovat soubor na str 135*
Zkontrolujte velikost souboru	*FileSize - Vyhledat velikost souboru na str 1155*
Zkontrolujte velikost souborového systému	*FSSize - Vyhledat velikost souborového systému na str 1161*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.94 IsMechUnitActive - Je mechanická jednotka aktivní

Použití

`IsMechUnitActive` (*Is Mechanical Unit Active*) se používá pro kontrolu, jestli je mechanická jednotka aktivována nebo nikoliv.

Základní příklady

Následující příklad názorně ukazuje funkci`IsMechUnitActive`.

Příklad 1

```
IF IsMechUnitActive(SpotWeldGun)
  CloseGun SpotWeldGun;
```

Jestliže mechanická jednotka `SpotWeldGun` je aktivní, rutina `CloseGun` bude vyvolána, ve které je pistole zavřena.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Funkce vrací:

- `TRUE`, jestliže mechanická jednotka je aktivní
- `FALSE`, jestliže mechanická jednotka je neaktivní

Argumenty

```
IsMechUnitActive ( MechUnit )
```

`MechUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky.

Syntaxe

```
IsMechUnitActive '('
  [ MechUnit ':='] < variable (VAR) of mecunit> ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Aktivace mechanických jednotek	*ActUnit - Aktivuje mechanickou jednotku na str 24*
Deaktivace mechanických jednotek	*DeactUnit - Deaktivuje mechanickou jednotku na str 150*
Mechanické jednotky	*mecunit - Mechanická jednotka na str 1531*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.95 IsPers - Je perzistent

Použití

`IsPers` se používá pro testování, jestli je datový objekt perzistentní proměnnou nebo nikoliv.

Základní příklady

Následující příklad názorně ukazuje funkci`IsPers`.

Příklad 1

```
PROC procedure1 (INOUT num parameter1)
  IF IsVar(parameter1) THEN
    ! For this call reference to a variable
    ...
  ELSEIF IsPers(parameter1) THEN
    ! For this call reference to a persistent variable
    ...
  ELSE
    ! Should not happen
    EXIT;
  ENDIF
ENDPROC
```

Procedura `procedure1` bude provádět různé činnosti podle toho, jestli aktuální parametr `parameter1` je proměnnou nebo perzistentní proměnnou.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

TRUE, jestliže testovaný aktuální parametr `INOUT` je perzistentní proměnnou.
FALSE, jestliže testovaný aktuální parametr `INOUT` není perzistentní proměnnou.

Argumenty

```
IsPers (DatObj)
```

`DatObj()`

Data Object

Datový typ: jakýkoliv typ

Jméno formálního parametru INOUT.

Syntaxe

```
IsPers '('
  [ DatObj ':=' ] < var or pers (INOUT) of any type > ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Test, jestli se jedná o proměnnou	*IsVar - Je proměnná na str 1221*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Typy parametrů (přístupové režimy)	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Rutiny*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.96 IsStopMoveAct - Je příznak zastavení pohybu aktivní

Použití

`IsStopMoveAct` se používá pro získání statutu příznaků pro zastavení pohybu pro aktuální nebo připojenou pohybovou úlohu.

Základní příklady

Následující příklady názorně ukazují funkci`IsStopMoveAct`.

Příklad 1

```
stopflag2:= IsStopMoveAct(\FromNonMoveTask);
```

`stopflag2` bude TRUE, jestliže příznak zastavení pohybu od nepohybových úloh je nastaven v aktuální nebo připojené pohybové úloze, jinak bude FALSE.

Příklad 2

```
IF IsStopMoveAct(\FromMoveTask) THEN
  StartMove;
ENDIF
```

Jestliže příznak zastavení pohybu od pohybové úlohy je nastaven v aktuální pohybové úloze, bude resetován instrukcí `StartMove`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Vrácená hodnota bude TRUE, jestliže je nastaven zvolený příznak zastavení pohybu, jinak bude vrácená hodnota FALSE.

Argumenty

```
IsStopMoveAct ( [\FromMoveTask] | [\FromNonMoveTask] )
```

`[\FromMoveTask]`

Datový typ: `switch`

`FromMoveTask` se používá pro získání statutu příznaku zastavení pohybu typu privátní pohybové úlohy.

Tento typ příznaku zastavení pohybu může být nastaven pouze od:

- Samotné pohybové úlohy s instrukcí `StopMove`
- Po opuštění úrovně `RestoPath` v programu
- Při vykonávání v asynchronním chybovém handleru pro procesní nebo pohybové chyby před jakoukoliv `StorePath` a po jakékoliv `RestoPath`

`[\FromNonMoveTask]`

Datový typ: `switch`

`FromNonMoveTask` se používá k získání statutu příznaku zastavení pohybu typu jakýchkoliv nepohybových úloh. Tento typ příznaku zastavení pohybu může být nastaven pouze jakoukoliv nepohybovou úlohou v připojených nebo všech pohybových úlohách s instrukcí `StopMove`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
IsStopMoveAct '('
  ['\' FromMoveTask]
  | ['\' FromNonMoveTask] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Zastavit pohyb robotu	*StopMove - Zastavuje pohyby robotu na str 736*
Restartovat pohyb robotu	*StartMove - Znovu spouští pohyb robotu na str 707*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.97 IsStopStateEvent - Otestovat, jestli se ukazatel programu posunul

Použití

`IsStopStateEvent` vrací informace o pohybu ukazatele programu (PP) v aktuální programové úloze.

Základní příklady

Následující příklad názorně ukazuje funkci `IsStopStateEvent`.

Příklad 1

```
IF IsStopStateEvent (\PPMoved) = TRUE THEN
  ! PP has been moved during the last program stop
ELSE
  ! PP has not been moved during the last program stop
ENDIF

IF IsStopStateEvent (\PPToMain) THEN
  ! PP has been moved to main routine during the last program stop
ENDIF
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Status jestli a jak byl PP posunut během posledního stop stavu.

`TRUE`, jestliže PP byl posunut během posledního zastavení.

`FALSE`, jestliže PP nebyl posunut během posledního zastavení.

Jestliže PP byl posunut k main rutině, oba `\PPMoved` a `\PPToMain` vrátí `TRUE`.

Jestliže PP byl posunut k rutině, oba `\PPMoved` a `\PPToMain`vrátí `TRUE`.

Jestliže PP byl posunut v seznamu rutiny, `\PPMoved` vrátí `TRUE` a `\PPToMain`vrátí `FALSE`.

Po volání servisní rutiny (udržujte kontext vykonávání v hlavní programové sekvenci) `\PPMove` vrátí `FALSE` a `\PPToMain` vrátí `FALSE`.

Argumenty

```
IsStopStateEvent ([\PPMoved] | [\PPToMain])
```

`[ \PPMoved ]`

Datový typ: `switch`

Test, jestli PP byl posunut.

`[ \PPToMain ]`

Datový typ: `switch`

Otestovat, jestli PP byl posunut k main nebo k rutině.

Omezení

Tuto funkci není možné používat ve většině případů během vykonávání dopředu nebo dozadu, protože systém je ve stop stavu mezi každým jednotlivým krokem.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
IsStopStateEvent '('
  ['\' PPMoved] | ['\ä PPToMain] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Provádění vlastních instrukcí	*Technická referenční příručka - Přehled RAPID*, sekce - *Programování off-line - Provádění vašich vlastních instrukcí*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.98 IsSyncMoveOn - Testovat synchronizovaný režim

Použití

`IsSyncMoveOn` se používá k testování, jestli aktuální programová úloha typu `Motion Task` je v režimu synchronizovaného pohybu nebo nikoliv.

Je také možné z některých `Non Motion Task` testovat, jestli připojená `Motion Task` je v režimu synchronizovaného pohybu nebo nikoliv. Systémový parametr *Controller/Tasks/Use Mechanical Unit Group* definuje připojenou `Motion Task`.

Když je vykonávána `Motion Task` na `StorePath` úrovni, `IsSyncMoveOn` bude testovat, jestli úloha je v synchronizovaném režimu na této úrovni, nezávisle na synchronizovaném režimu na původní úrovni.

Instrukce `IsSyncMoveOn` se obvykle používá v systému *MultiMove* s doplňkem *Coordinated Robots*, ale může být použita v každém systému a v každé programové úloze.

Základní příklady

Následující příklad názorně ukazuje funkci`IsSyncMoveOn`.

Příklad 1

Příklad programu v úloze `T_ROB1`

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;

PROC main()
  ...
  MoveL p_zone, vmax, z50, tcp1;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, tcp1;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, tcp1 \WOBJ:= rob2_obj;
  MoveL * \ID:=20, v100, fine, tcp1 \WOBJ:= rob2_obj;
  SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
ENDPROC
```

Příklad programu v úloze `BCK1`

```
PROC main()
  ...
  IF IsSyncMoveOn() THEN
    ! Connected Motion Task is in synchronized movement mode
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
    ELSE
      ! Connected Motion Task is in independent mode
    ENDIF
    ...
  ENDPROC
```

V době vykonávání `IsSyncMoveOn` v úloze v pozadí `BCK1` testujeme, jestli připojená pohybová úloha je v té chvíli v režimu synchronizovaného pohybu nebo nikoliv.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže aktuální nebo připojená programová úloha je v té chvíli v režimu synchronizovaného pohybu, jinak `FALSE`.

Vykonávání programu

Testovat, jestli aktuální nebo připojená programová úloha je v té chvíli v režimu synchronizovaného pohybu nebo nikoliv. Jestliže `MotionTask` je vykonávána na `StorePath level`, potom `SyncMoveOn` bude testovat, jestli úloha je v synchronizovaném pohybu na `StorePath level`, nikoliv na původní úrovni.

Syntaxe

```
IsSyncMoveOn '(' ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Určit spolupracující programové úlohy	*tasks - Programové úlohy RAPID na str 1607*
Identita pro bod synchronizace	*syncident - Identita pro bod synchronizace na str 1603*
Spustit koordinované synchronizované pohyby	*SyncMoveOn - Spustit koordinované synchronizované pohyby na str 758*
Ukončit koordinované synchronizované pohyby	*SyncMoveOff - Ukončit koordinované synchronizované pohyby na str 752*
Nastavit nezávislé pohyby	*SyncMoveUndo - Nastavit nezávislé pohyby na str 768*
Uložit cestu a vykonávat na nové úrovni	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.99 IsSysId - Otestovat systémovou identitu

Použití

`IsSysId` (*System Identity*) se může použít k testování systémové identity pomocí systémového sériového čísla.

Základní příklady

Následující příklad názorně ukazuje funkci`IsSysId`.

Příklad 1

```
IF NOT IsSysId("6400-1234") THEN
  ErrWrite "System identity fault", "Faulty system identity for
       this program";
  EXIT;
ENDIF
```

Program je vytvořen pro speciální robotický systém se sériovým číslem `6400-1234` a nemůže být použit jiným robotickým systémem.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE` = Sériové číslo robotického systému je stejné, jak bylo určeno v testu.

`FALSE` = Sériové číslo robotického systému není stejné, jak bylo určeno v testu.

Argumenty

```
IsSysId ( SystemId)
```

`SystemId`

Datový typ: `string`

Sériové číslo robotického systému označující systémovou identitu.

Syntaxe

```
IsSysId '('
  [ SystemId ':=' ] < expression (IN) of string> ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Přečíst systémovou informaci	*GetSysInfo - Získat informace o systému na str 1182*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.100 IsVar - Je proměnná

Použití

`IsVar` se používá pro testování, jestli je datový objekt proměnnou nebo nikoliv.

Základní příklady

Následující příklad názorně ukazuje funkci`IsVar`.

Příklad 1

```
PROC procedure1 (INOUT num parameter1)
  IF IsVAR(parameter1) THEN
    ! For this call reference to a variable
    ...
  ELSEIF IsPers(parameter1) THEN
    ! For this call reference to a persistent variable
    ...
  ELSE
    ! Should not happen
    EXIT;
  ENDIF
ENDPROC
```

Procedura `procedure1` bude provádět různé činnosti podle toho, jestli aktuální parametr `parameter1` je proměnnou nebo perzistentní proměnnou.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

TRUE, jestliže testovaný aktuální parametr INOUT je proměnnou. FALSE, jestliže testovaný aktuální parametr INOUT není proměnnou.

Argumenty

```
IsVar (DatObj)
```

`DatObj`

Data Object

Datový typ: jakýkoliv typ

Jméno formálního parametru `INOUT`.

Syntaxe

```
IsVar '('
  [ DatObj ':=' ] < var or pers (INOUT) of any type > ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Otestovat, jestli se jedná o perzistent	*IsPers - Je perzistent na str 1212*
Typy parametrů (přístupové režimy)	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Rutiny*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.101 MaxRobSpeed - Max rychlost robotu

Použití

`MaxRobSpeed` (*Maximum Robot Speed*) vrací max. rychlost TCP pro používaný typ robotu.

Základní příklady

Následující příklad názorně ukazuje funkci`MaxRobSpeed`.

Příklad 1

```
TPWrite "Max. TCP speed in mm/s for my robot="\Num:=MaxRobSpeed();
```

Zpráva `Max. TCP speed in mm/s for my robot = 5000` je zapsána na FlexPendant.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Vrátit max TCP rychlost v mm/s pro používaný typ robotu a normální praktické hodnoty TCP.

Extrémně velké hodnoty TCP se používají v nástrojovém rámci, každý by si měl vytvořit svá vlastní speeddata s větší rychlostí TCP, než jaká je vrácena od `MaxRobSpeed`.

Syntaxe

```
MaxRobSpeed '(' ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Definice rychlosti	*speeddata - Rychlostní data na str 1586*
Definice max rychlosti	*VelSet - Mění naprogramovanou rychlost na str 913*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.102 MirPos - Zrcadlení pozice

Použití

`MirPos` (*Mirror Position*) se používá k zrcadlení překladu a rotačních částí pozice.

Základní příklady

Následující příklad názorně ukazuje funkci`MirPos`.

Příklad 1

```
CONST robtarget p1:= [...];
VAR robtarget p2;
PERS wobjdata mirror:= [...];
...
p2 := MirPos(p1, mirror);
```

`p1` je robtarget ukládající pozici robotu a orientace nástroje. Tato pozice je zrcadlena v rovině xy rámce definovaného od `mirror`, relativně ke světovému souřadnému systému. Výsledkem jsou nová data robtarget, která jsou uložena do `p2`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `robtarget`

Nová pozice, která je zrcadlovou pozicí vstupní pozice.

Argumenty

```
MirPos (Point MirPlane [\WObj] [\MirY])
```

`Point`

Datový typ: `robtarget`

Vstupní pozice robotu. Orientační část této pozice definuje aktuální orientaci souřadného systému nástroje.

`MirPlane`

Mirror Plane

Datový typ: `wobjdata`

Data pracovního objektu definující rovinu zrcadla. Rovina zrcadla je rovina xy rámce objektu definovaného v `MirPlane`. Umístění rámce objektu je definováno ve vztahu s uživatelskému rámci (definováno také v `MirPlane`), který je obratem definován ve vztahu ke světovému rámci.

`[\WObj]`

Work Object

Datový typ: `wobjdata`

Data pracovního objektu definující rámec objektu a uživatelský rámec ve vztahu, ke kterému je definována vstupní pozice *Point*. Jestliže tento argument je vynechán, pozice je definována ve vztahu ke světovému souřadnému systému.

UPOZORNENÍ!

Jestliže je vytvořena pozice s aktivním pracovním objektem, tento pracovní objekt musí být odkazován v argumentu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.102 MirPos - Zrcadlení pozice

RobotWare - OS

Pokračování

`[\MirY]`

Mirror Y

Datový typ: `switch`

Jestliže tento přepínač je vynechán, což je výchozí chování, rámec nástroje bude zrcadlen s ohledem na osu x a osu z. Jestliže přepínač je určen, rámec nástroje bude zrcadlen s ohledem na osu y a osu z.

Omezení

Neprovádí se žádná nová kalkulace části konfigurace robotu vstupních dat robtarget.

Jestliže je použit rámec souřadnice, koordinovaná jednotka musí být situována ve stejné úloze jako robot.

Syntaxe

```
MirPos '('
  [ Point ':=' ] < expression (IN) of robtarget> ','
  [ MirPlane ':='] <expression (IN) of wobjdata> ','
  ['\' WObj ':=' <expression (IN) of wobjdata> ]
  ['\' MirY ] ')'
```

Funkce s vrácenou hodnotou datového typu `robtarget`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Matematika*
Poziční data	*robtarget - Poziční data na str 1572*
Data pracovního objektu	*wobjdata - Data pracovního objektu na str 1634*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.103 MOD - Vyhodnocuje modulo celého čísla

Použití

MOD je podmíněný výraz, který se používá k vyhodnocení modulo, zbytku dělení celých čísel.

Základní příklady

Následující příklady názorně ukazují funkciMOD.

Příklad 1

```
reg1 := 14 MOD 4;
```

Vrácená hodnota je 2, protože14 děleno 4 dává modulo 2.

Příklad 2

```
VAR dnum mydnum1 := 11;
VAR dnum mydnum2 := 5;
VAR dnum mydnum3;
...
mydnum3 := mydnum1 MOD mydnum2;
```

Vrácená hodnota je 1, protože11 děleno 5 dává modulo 2.

Vratná hodnota (Vrátit hodnotu)

Datový typ: numdnum

Vrací modulo, zbytek z dělení celých čísel.

Syntaxe

```
<expression of num> MOD <expression of num>
```

Funkce s vrácenou hodnotou datového typu num.

```
<expression of dnum> MOD <expression of dnum>
```

Funkce s vrácenou hodnotou datového typu dnum.

Související informace

Pro informace o	Viz
num - Numerické hodnoty	*num - Numerické hodnoty na str 1538*
dnum - Dvojité numerické hodnoty	*dnum - Dvojité numerické hodnoty na str 1485*
DIV	*DIV - Vyhodnocuje dělení celého čísla na str 1138*
Výrazy	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.104 ModExist - Zkontrolujte, jestli existuje programový modul

Použití

`ModExist` (*Module Exist*) se používá ke kontrole, jestli daný modul existuje v programové úloze nebo nikoliv.

Hledání je provedeno nejdříve u načtených modulů a potom, není-li žádný nalezen, u instalovaných modulů.

Základní příklady

Následující příklad názorně ukazuje funkci`ModExist`.

Příklad 1

```
VAR bool mod_exist;
mod_exist:=ModExist ("MyModule");
```

Jestliže modul `MyModule` existuje v úloze, funkce vrátí `TRUE`. Jestliže nikoliv, funkce vrátí `FALSE`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

TRUE, jestliže modul byl nalezen, FALSE, jestliže nikoliv.

Argumenty

```
ModExist (ModuleName)
```

`ModuleName`

Datový typ: `string`

Jméno modulu, který se má hledat.

Syntaxe

```
ModExist '('
  [ ModuleName ':=' ] < expression (IN) of string > ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Vyhledat čas modifikace načteného modulu	*ModTimeDnum - Získat čas modifikace souboru pro načtený modul na str 1227*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.105 ModTimeDnum - Získat čas modifikace souboru pro načtený modul

Použití

`ModTimeDnum` (*Modify Time*) se používá k vyhledání času poslední modifikace souboru pro načtený modul. Modul je určen svým jménem a musí být v paměti úlohy. Čas je měřen v sekundách od 00:00:00 GMT, 1. ledna 1970. Čas je vrácen jako `dnum` a volitelně také jako `stringdig`.

Základní příklady

Následující příklad názorně ukazuje funkci`ModTimeDnum`.

Viz také *Další příklady na str 1227*.

Příklad 1

```
MODULE mymod
  VAR dnum mytime;
  PROC printMyTime()
    mytime := ModTimeDnum("mymod");
    TPWrite "My time is "+ValToStr(mytime);
  ENDPROC
ENDMODULE
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Čas měřený v sekundách od 00:00:00 GMT, 1. ledna 1970.

Argumenty

```
ModTimeDnum ( Object [\StrDig] )
```

`Object`

Datový typ: `string`

Jméno modulu.

`[\StrDig]`

String Digit

Datový typ: `stringdig`

Získání času načtení modulu v reprezentaci `stringdig`.

Vykonávání programu

Tato funkce vrací numerickou hodnotu, která určuje poslední čas, kdy byl soubor modifikován před načtením jako programový modul do systému.

Další příklady

Více příkladů funkce `ModTimeDnum` je názorně uvedeno dole.

Příklad 1

```
IF FileTimeDnum ("HOME:/mymod.mod" \ModifyTime) > ModTimeDnum
     ("mymod") THEN
  UnLoad "HOME:/mymod.mod";
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.105 ModTimeDnum - Získat čas modifikace souboru pro načtený modul

Pokračování

```
    Load \Dynamic, "HOME:/mymod.mod";
  ENDIF
```

Tento program znovu načítá modul, jestliže zdrojový soubor je novější. Používá `ModTimeDnum` k vyhledání poslední modifikace určeného modulu a porovnává ji s `FileTimeDnum ("HOME:/mymod.mod" \ModifyTime)` u zdroje. Potom, jestliže zdroj je novější, program stáhne a znovu načte modul.

Řešení chyb

Jestliže modul s určeným jménem není v programové úloze, potom je systémová proměnná `ERRNO` nastavena na `ERR_MOD_NOT_LOADED`. Tato chyba může být potom ošetřena v chybovém handleru.

Omezení

Tato funkce vždy vrátí `0`, jestliže je použita na modulu, který je zakódován nebo instalován sdíleně.

Syntaxe

```
ModTimeDnum '('
  [ Object ':=' ] < expression (IN) of string>
  [ '\' StrDig ':=' < variable (VAR) of stringdig> ] ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Vyhledat časovou informaci o souboru	*FileTimeDnum - Získat časovou informaci o souboru na str 1158*
Řetězec pouze s číslicemi	*stringdig - Řetězec pouze s číslicemi na str 1598*
Porovnat dva řetězce pouze s číslicemi	*StrDigCmp - Porovnat dva řetězce pouze s číslicemi na str 1336*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.106 MotionPlannerNo - Získat číslo připojeného plánovače pohybů

Použití

`MotionPlannerNo` **vrací číslo připojeného plánovače pohybů. Při vykonávání** `MotionPlannerNo` **v pohybové úloze vrací číslo svého plánovače. Jinak při vykonávání** `MotionPlannerNo` **v nepohybové úloze vrací číslo připojeného plánovače pohybů podle nastavení v systémových parametrech.**

Základní příklady

Následující příklad názorně ukazuje funkci`MotionPlannerNo`.

Příklad 1

```
!Motion task T_ROB1
PERS string buffer{6} := ["", "", "", "", "", ""];
VAR num motion_planner;

PROC main()
  ...
  MoveL point, v1000, fine, tcp1;
  motion_planner := MotionPlannerNo();
  buffer{motion_planner} := "READY";
  ...
ENDPROC

!Background task BCK1
PERS string buffer{6};
VAR num motion_planner;
VAR string status;

PROC main()
  ...
  motion_planner := MotionPlannerNo();
  status := buffer{motion_planner};
  ...
ENDPROC

!Motion T_ROB2
PERS string buffer{6};
VAR num motion_planner;

PROC main()
  ...
  MoveL point, v1000, fine, tcp1;
  motion_planner := MotionPlannerNo();
  buffer{motion_planner} := "READY";
  ...
ENDPROC

!Background task BCK2
PERS string buffer{6};
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.106 MotionPlannerNo - Získat číslo připojeného plánovače pohybů

RobotWare - OS

Pokračování

```
VAR num motion_planner;
VAR string status;

PROC main()
  ...
  motion_planner := MotionPlannerNo();
  status := buffer{motion_planner};
  ...
ENDPROC
```

Použijte funkci `MotionPlannerNo` ke zjištění, které číslo plánovače pohybů je připojeno k úloze. Přesně stejný kód může být implementován do všech pohybových úloh a úloh v pozadí. Potom může každá úloha v pozadí kontrolovat status pro svoji připojenou pohybovou úlohu.

Vratná hodnota (Vrátit hodnotu)

Datový typ:`num`

Číslo připojeného plánovače pohybů. U nepohybových úloh bude vráceno číslo plánovače pohybů přidružené mechanické jednotky.

Rozsah vrácené hodnoty je 1 ... 6.

Syntaxe

```
MotionPlannerNo '(' ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Určit spolupracující programové úlohy	*Technická referenční příručka - Systémové parametry*, sekce *Controller - Task*.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.107 NonMotionMode - Přečíst nepohybový prováděcí režim

Použití

`NonMotionMode` (*Non-Motion Execution Mode*) se používá ke čtení aktuálního nepohybového prováděcího režimu programové úlohy. Nepohybový prováděcí režim je zvolen nebo odstraněn z FlexPendantu pod nabídkou *ABB\Ovládací panel\Dohled*.

Základní příklady

Následující příklad názorně ukazuje funkci`NonMotionMode`.

Příklad 1

```
IF NonMotionMode() =TRUE THEN
  ...
ENDIF
```

Sekce programu je vykonána pouze když robot je v nepohybovém prováděcím režimu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Aktuální nepohybový režim, jak je definován v tabulce dole.

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
0	FALSE	Nepohybové vykonávání se nepoužívá
1	TRUE	Nepohybové vykonávání se používá

Argumenty

```
NonMotionMode ( [ \Main] )
```

`[ \Main ]`

Datový typ: `switch`

Vrátit aktuální běžící režim pro připojenou pohybovou úlohu. Používá se v multitaskingovém systému k získání aktuálního běžícího režimu pro aktuální úlohu, jestliže je to pohybová úloha nebo připojená pohybová úloha, jestliže funkce `NonMotionMode` je vykonávána v nepohybové úloze.

Jestliže tento argument je vynechán, vrácená hodnota vždy zrcadlí aktuální běžící režim pro programovou úlohu, která vykonává funkci `NonMotionMode`.

Všimněte si, že režim vykonávání je spojen se systémem a nikoliv s kteroukoliv úlohou. To znamená, že všechny úlohy v systému budou dávat stejnou vrácenou hodnotu od `NonMotionMode`.

Syntaxe

```
NonMotionMode '(' ['\' Main] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.107 NonMotionMode - Přečíst nepohybový prováděcí režim

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Čtení provozního režimu	*OpMode - Přečíst provozní režim na str 1241*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.108 NOT - Invertuje logickou hodnotu

Použití

`NOT` je podmíněný výraz, který se používá k invertování logické hodnoty (true/false).

Základní příklady

Následující příklady názorně ukazují podmíněný výraz`NOT`.

Příklad 1

```
VAR bool mybool;
mybool := NOT mybool;
```

Jestliže `mybool` je `TRUE`, vrácená hodnota je FALSE.

Jestliže `mybool` je `FALSE`, vrácená hodnota je TRUE.

Příklad 2

```
VAR bool a;
VAR bool b;
VAR bool c;
...
c := a AND (NOT b);
```

Vrácená hodnota `c` je TRUE, jestliže `a` je `TRUE` a `b` je `FALSE`

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Vrací invertovanou hodnotu.

Syntaxe

```
NOT <logical term>
```

Související informace

Pro informace o	Viz
AND	*AND - Vyhodnocuje logickou hodnotu na str 1035*
OR	*OR - Vyhodnocuje logickou hodnotu na str 1242*
XOR	*XOR - Vyhodnocuje logickou hodnotu na str 1435*
Výrazy	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.109 NOrient - Normalizovat orientaci

Použití

`NOrient` (*Normalize Orientation*) se používá k normalizaci nenormalizované orientace (kvaternion).

Popis

Orientace musí být normalizována, tj. součet čtverců se musí rovnat 1:

$$q_1^2+q_2^2+q_3^2+q_4^2=1$$

xx0500002452

Jestliže orientace je mírně nenormalizována, je možné ji normalizovat. Normalizační chyba je absolutní hodnota součtu čtverců komponentů orientace. Orientace je považována za místně nenormalizovanou, jestliže normalizační chyba je větší než 0,00001 a menší než 0,1. Jestliže normalizační chyba je větší než 0,1, orientace je nepoužitelná.

$$\mathrm{ABS}(\sqrt{q_1^2+q_2^2+q_3^2+q_4^2}-1)=\mathrm{normerr}$$

xx0500002453

normerr > 0.1Nepoužitelné

normerr > 0.00001 AND normerr <= 0.1Mírně nenormalizováno

normerr <= 0.00001Normalizováno

Základní příklady

Následující příklad názorně ukazuje funkci`NOrient`.

Příklad 1

Máme mírně nenormalizovanou pozici (0.707170, 0, 0, 0.707170)

$$\mathrm{ABS}(\sqrt{0{,}707170^2+0^2+0^2+0{,}707170^2}-1) = 0{,}0000894$$

$$0{,}0000894 > 0{,}00001 \Rightarrow \text{unnormalized}$$

xx0500002451

```
VAR orient unnormorient := [0.707170, 0, 0, 0.707170];
VAR orient normorient;
...
...
normorient := NOrient(unnormorient);
```

Normalizace orientace (0.707170, 0, 0, 0.707170) dostává (0.707107, 0, 0, 0.707107).

Vratná hodnota (Vrátit hodnotu)

Datový typ: `orient`

Normalizovaná prientace.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Argumenty

```
NOrient (Rotation)
```

Otočení

Datový typ: `orient`

Orientace, která by měla být normalizována.

Syntaxe

```
NOrient '('
  [Rotation ':='] <expression (IN) of orient> ')'
```

Funkce s vrácenou hodnotou datového typu `orient`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Matematika*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.110 NumToDnum - Převádí num na dnum

Použití

`NumToDnum` **převádí** `num` **na** `dnum`.

Základní příklady

Následující příklad názorně ukazuje funkci`NumToDnum`.

Příklad 1

```
VAR num mynum:=55;
VAR dnum mydnum:=0;
mydnum:=NumToDnum(mynum);
```

Num hodnota 55 je vrácena funkcí jako `dnum` **hodnota 55.**

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Vrácená hodnota typu `dnum` **bude mít stejnou hodnotu jako vstupní hodnota typu** `num`.

Argumenty

```
NumToDnum (Value)
```

`Value`

Datový typ: `num`

Numerická hodnota, která bude převedena.

Syntaxe

```
NumToDnum '('
  [ Value ':=' ] < expression (IN) of num > ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Datový typ `Num`	*num - Numerické hodnoty na str 1538*
Datový typ `Dnum`	*dnum - Dvojité numerické hodnoty na str 1485*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.111 NumToStr - Převádí numerickou hodnotu na řetězec

Použití

`NumToStr` (*Numeric To String*) se používá k převodu numerické hodnoty na řetězec.

Základní příklady

Následující příklady názorně ukazují funkci`NumToStr`.

Příklad 1

```
VAR string str;
str := NumToStr(0.38521,3);
```

Proměnné `str` je dána hodnota `"0.385"`.

Příklad 2

```
reg1 := 0.38521;
str := NumToStr(reg1, 2\Exp);
```

Proměnné `str` je dána hodnota `"3.85E-01"`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Numerická hodnota převedená na řetězec s určeným počtem desetinných čísel, s exponentem, jestliže je to požadováno. Numerická hodnota je podle potřeby zaokrouhlena. Desetinná tečka (čárka) je potlačena, jestliže nejsou obsažena žádná desetinná místa.

Argumenty

```
NumToStr (Val Dec [\Exp])
```

`Val`

Value

Datový typ: `num`

Numerická hodnota, která bude převedena.

`Dec`

Decimals

Datový typ: `num`

Počet desetinných míst. Počet desetinných míst nesmí být záporný nebo větší než dostupná řádová přesnost pro numerické hodnoty.

`[\Exp]`

Exponent

Datový typ: `switch`

Použití exponentu ve vrácené hodnotě.

Syntaxe

```
NumToStr '('
  [ Val ':=' ] <expression (IN) of num>
  [ Dec ':=' ] <expression (IN) of num>
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.111 NumToStr - Převádí numerickou hodnotu na řetězec

RobotWare - OS

Pokračování

```
['\' Exp ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Řetězcové funkce*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Základní prvky*
Převést numerickou hodnotu dnum na řetězec	*DnumToStr - Převádí numerickou hodnotu na řetězec na str 1141*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.112 Offs - Posunuje pozici robotu

Použití

`Offs` se používá pro přidání ofsetu v souřadném systému objektu k pozici robotu.

Základní příklady

Následující příklady názorně ukazují funkci`Offs`.

Viz také *Další příklady na str 1239*.

Příklad 1

```
MoveL Offs(p2, 0, 0, 10), v1000, z50, tool1;
```

Robot je posunut k bodu `10` mm od pozice `p2` (ve směru z).

Příklad 2

```
p1 := Offs (p1, 5, 10, 15);
```

Pozice robotu `p1` je posunuta `5` mm ve směru x, `10` mm ve směru y a `15` mm ve směru z.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `robtarget`

Posunutá poziční data.

Argumenty

```
Offs (Point XOffset YOffset ZOffset)
```

`Point`

Datový typ: `robtarget`

Poziční data, která budou posunuta.

`XOffset`

Datový typ: `num`

Posunutí ve směru x v souřadném systému objektu.

`YOffset`

Datový typ: `num`

Posunutí ve směru y v souřadném systému objektu.

`ZOffset`

Datový typ: `num`

Posunutí ve směru z v souřadném systému objektu.

Další příklady

Více příkladů funkce `Offs` je názorně uvedeno dole.

Příklad 1

```
PROC pallet (num row, num column, num distance, PERS tooldata tool,
     PERS wobjdata wobj)
VAR robtarget palletpos:=[[0, 0, 0], [1, 0, 0, 0], [0, 0, 0, 0],
     [9E9, 9E9, 9E9, 9E9, 9E9, 9E9]];
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.112 Offs - Posunuje pozici robotu

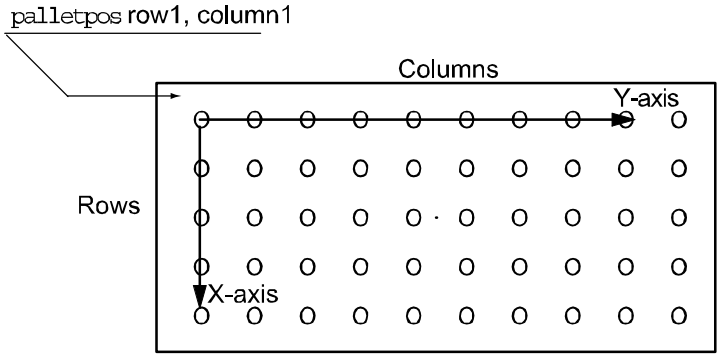
RobotWare - OS

Pokračování

```
palettpos := Offs (palettpos, (row-1)*distance, (column-1)*distance,
     0);
MoveL palettpos, v100, fine, tool\WObj:=wobj;
ENDPROC
```

Rutina pro nabrání kusů z palety je provedena. Každá paleta je definována jako pracovní objekt (viz obrázek dole). Kus, který bude nabrán (řada a sloupec) a vzdálenost mezi kusy jsou dány jako vstupní parametry. Přírůstek indexu řady a sloupce se provádí mimo rutinu.

Obrázek ukazuje pozici a orientace palety je určena definováním pracovního objektu.



xx050002300_

Syntaxe

```
Offs '('
  [Point ':='] <expression (IN) of robtarget> ','
  [XOffset ':='] <expression (IN) of num> ','
  [YOffset ':='] <expression (IN) of num> ','
  [ZOffset ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `robtarget`.

Související informace

Pro informace o	Viz
Poziční data	*robtarget - Poziční data na str 1572*
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Matematika*
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.113 OpMode - Přečíst provozní režim

Použití

`OpMode` *Operating Mode* se používá pro čtení aktuálního provozního režimu systému.

Základní příklady

Následující příklad názorně ukazuje funkci `OpMode`.

Příklad 1

```
TEST OpMode()
CASE OP_AUTO:
  ...
CASE OP_MAN_PROG:
  ...
CASE OP_MAN_TEST:
  ...
DEFAULT:
  ...
ENDTEST
```

Různé programové sekce jsou vykonávány podle aktuálního provozního režimu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `symnum`

Aktuální provozní režim, jak je definován v tabulce dole.

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
0	OP_UNDEF	Nedefinovaný provozní režim
1	OP_AUTO	Automatizovaný provozní režim
2	OP_MAN_PROG	Ruční provozní režim max. 250 mm/s
3	OP_MAN_TEST	Ruční provozní režim s plnou rychlostí, 100 %

Syntaxe

```
OpMode '(' ')'
```

Funkce s vrácenou hodnotou datového typu `symnum`.

Související informace

Pro informace o	Viz
Různé provozní režimy	*Návod k použití - IRC5 s jednotkou FlexPendant*
Čtení běžícího režimu	*RunMode - Přečíst provozní režim na str 1311*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.114 OR - Vyhodnocuje logickou hodnotu

Použití

OR je podmíněný výraz, který se používá k hodnocení logické hodnoty (true/false).

Základní příklady

Následující příklady názorně ukazují funkciOR.

Příklad 1

```
VAR num a;
VAR num b;
VAR bool c;
...
c := a>5 OR b=3;
```

Vrácená hodnota c je TRUE, jestliže a je větší než 5 a b je rovno 3. Jinak je vrácená hodnota FALSE.

Příklad 2

```
VAR num mynum;
VAR string mystring;
VAR bool mybool;
VAR bool result;
...
result := mystring="Hello" OR mynum<15 AND mybool;
```

Vrácená hodnota result je TRUE, jestliže mystring je "Hello". Nebo kdyžmynum jsou menší než 15 a mybool je TRUE. Jinak je vrácená hodnota FALSE.

Příkaz AND je vyhodnocen jako první, potom příkaz OR. Toto je ilustrováno závorkami v řádce dole.

```
result := mystring="Hello" OR (mynum<15 AND mybool);
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: bool

Vrácená hodnota je TRUE, jestliže jeden nebo oba podmíněné výrazy jsou správné, jinak je vrácená hodnota FALSE.

Syntaxe

```
<expression of bool> OR <expression of bool>
```

Funkce s vrácenou hodnotou datového typu bool.

Související informace

Pro informace o	Viz
AND	*AND - Vyhodnocuje logickou hodnotu na str 1035*
XOR	*XOR - Vyhodnocuje logickou hodnotu na str 1435*
NOT	*NOT - Invertuje logickou hodnotu na str 1233*
Výrazy	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.115 OrientZYX - Vytváří orient z úhlů euler

Použití

`OrientZYX` (*Orient from Euler ZYX angles*) se používá pro vytvoření proměnné typu orient z úhlů Euler.

Základní příklady

Následující příklad názorně ukazuje funkci`OrientZYX`.

Příklad 1

```
VAR num anglex;
VAR num angley;
VAR num anglez;
VAR pose object;
...
object.rot := OrientZYX(anglez, angley, anglex)
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `orient`

Orientace vytvořená z úhlů Euler.

Rotace budou provedeny v následujícím pořadí:

- rotace kolem osy z
- rotace kolem nové osy y,
- rotace kolem nové osy x.

Argumenty

```
OrientZYX (ZAngle YAngle XAngle)
```

`ZAngle`

Datový typ: `num`

Rotace, ve stupních, kolem osy Z.

`YAngle`

Datový typ: `num`

Rotace, ve stupních, kolem osy Y.

`XAngle`

Datový typ: `num`

Rotace, ve stupních, kolem osy X.

Rotace budou provedeny v následujícím pořadí:

- rotace kolem osy z
- rotace kolem nové osy y,
- rotace kolem nové osy x.

Syntaxe

```
OrientZYX '('
  [ZAngle ':='] <expression (IN) of num> ','
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.115 OrientZYX - Vytváří orient z úhlů euler

RobotWare - OS

Pokračování

```
[YAngle ':='] <expression (IN) of num> ','
[XAngle ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `orient`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Návod k použití - IRC5 s jednotkou FlexPendant*, sekce *Souhrn RAPID - Matematika*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.116 ORobT - Odstraňuje posun programu z pozice

Použití

`ORobT` (*Object Robot Target*) se používá k transformaci pozice robotu ze souřadného systému posunu programu do souřadného systému objektu a/nebo k odstranění ofsetu u externích os.

Základní příklady

Následující příklad názorně ukazuje funkci`ORobT`.

Viz také *Další příklady na str 1246*.

Příklad 1

```
VAR robtarget p10;
VAR robtarget p11;
VAR num wobj_diameter;

p10 := CRobT(\Tool:=tool1 \WObj:=wobj_diameter);
p11 := ORobT(p10);
```

Aktuální pozice robotu a externích os jsou uloženy v `p10` a `p11`. Hodnoty uložené v `p10` mají vztah k souřadnému systému ProgDisp/ExtOffs. Hodnoty uložené v `p11` mají vztah k souřadnému systému objektu bez jakéhokoliv posunu programu a jakéhokoliv ofsetu na externích osách.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `robtarget`

Transformovaná poziční data.

Argumenty

```
ORobT (OrgPoint [\InPDisp] | [\InEOffs])
```

`OrgPoint`

Original Point

Datový typ: `robtarget`

Původní bod, který bude transformován.

`[\InPDisp]`

In Program Displacement

Datový typ: `switch`

Vrací pozici TCP v souřadném systému `ProgDisp`, tj. odstraňuje pouze ofset externích os.

`[\InEOffs]`

In External Offset

Datový typ: `switch`

Vrací externí osy v souřadném systému ofsetu, tj. odstraňuje posun programu pouze u robotu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.116 ORobT - Odstraňuje posun programu z pozice

RobotWare - OS

Pokračování

Další příklady

Více příkladů jak používat instrukci `ORobT` je názorně uvedeno dole.

Příklad 1

```
p10 := ORobT(p10 \InEOffs );
```

Funkce `ORobT` odstraní jakýkoliv posun programu, který je aktivní, ponechá pozici TCP ve vztahu k souřadnému systému objektu. Externí osy zůstanou v souřadném systému ofsetu.

Příklad 2

```
p10 := ORobT(p10 \InPDisp );
```

Funkce `ORobT` odstraní jakýkoliv ofset externích os. Pozice TCP zůstane v souřadném systému `ProgDisp`.

Syntaxe

```
ORobT '('
  [ OrgPoint ':=' ] < expression (IN) of robtarget>
  ['\' InPDisp] | ['\' InEOffs] ')'
```

Funkce s vrácenou hodnotou datového typu `robtarget`.

Související informace

Pro informace o	Viz
Definice posunu programu pro robot	*PDispOn - Aktivuje posun programu na str 477* *PDispSet - Aktivuje posun programu pomocí známého rámce na str 481*
Definice ofsetu pro pomocné osy	*EOffsOn - Aktivuje ofset pro pomocné osy na str 197* *EOffsSet - Aktivuje ofset pro pomocné osy pomocí známých hodnot na str 199*
Souřadné systémy	*Návod k použití - IRC5 s jednotkou FlexPendant*, sekce*Principy pohybu a I/O - Souřadné systémy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.117 ParIdPosValid - Platná pozice robotu pro identifikaci parametru

Použití

`ParIdPosValid` (*Parameter Identification Position Valid*) kontroluje, jestli pozice robotu je platná pro identifikaci aktuálního parametru, jako je identifikace zátěže nástroje nebo užitečná zátěž.

Tuto instrukci je možné používat pouze v úloze `main` nebo v systému *MultiMove* v pohybových úlohách.

Základní příklady

Následující příklad názorně ukazuje funkci`ParIdPosValid`.

Příklad 1

```
VAR jointtarget joints;
VAR bool valid_joints{12};

! Check if valid robot type
IF ParIdRobValid(TOOL_LOAD_ID) <> ROB_LOAD_VAL THEN
  EXIT;
ENDIF
! Read the current joint angles
joints := CJointT();
! Check if valid robot position
IF ParIdPosValid (TOOL_LOAD_ID, joints, valid_joints) = TRUE THEN
  ! Valid position for load identification
  ! Continue with LoadId
  ...
ELSE
  ! Not valid position for one or several axes for load
  ! identification
  ! Move the robot to the output data given in variable joints
  ! and do ParIdPosValid once again
  ...
ENDIF
```

Před prováděním identifikace zátěže nástroje zkontrolujte, jestli pozice robotu je platná.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže pozice robotu je platná pro identifikaci aktuálního parametru.

`FALSE`, jestliže pozice robotu není platná pro identifikaci aktuálního parametru.

Argumenty

```
ParIdPosValid (ParIdType Pos AxValid [\ConfAngle])
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.117 ParIdPosValid - Platná pozice robotu pro identifikaci parametru

RobotWare - OS

Pokračování

`ParIdType`

Datový typ: `paridnum`

Identifikace typu parametru, jak je definováno v tabulce dole.

Hodnota	Symbolická konstanta	Komentář
1	`TOOL_LOAD_ID`	Identifikovat zatížení nástroje
2	`PAY_LOAD_ID`	Identifikovat užitečnou zátěž (Ref. instrukce GripLoad)
3	`IRBP_K`	Identifikovat externí manipulátor IRBP K zátěž
4	`IRBP_L`	Identifikovat externí manipulátor IRBP L zátěž
4	`IRBP_C`	Identifikovat externí manipulátor IRBP C zátěž
4	`IRBP_C_INDEX`	Identifikovat externí manipulátor IRBP C_INDEX zátěž
4	`IRBP_T`	Identifikovat externí manipulátor IRBP T zátěž
5	`IRBP_R`	Identifikovat externí manipulátor IRBP R zátěž
6	`IRBP_A`	Identifikovat externí manipulátor IRBP A zátěž
6	`IRBP_B`	Identifikovat externí manipulátor IRBP B zátěž
6	`IRBP_D`	Identifikovat externí manipulátor IRBP D zátěž

`Pos`

Datový typ: `jointtarget`

Proměnná určuje úhly aktuálního spoje pro všechny osy robotu a externí osy. Proměnná je aktualizována od `ParIdPosValid` podle tabulky dole.

Vstupní hodnota spoje osy	Výstupní hodnota spoje osy
Platný	Nezměněno
Neplatný	Změněno na vhodnou hodnotu

`AxValid`

Datový typ: `bool`

Proměnná pole s 12 elementy odpovídající 6 osám robotu a 6 externím osám. Proměnná je aktualizována od `ParIdPosValid` podle tabulky dole.

Vstupní hodnota spoje osy v Pos	Výstupní status v AxValid
Platný	TRUE
Neplatný	FALSE

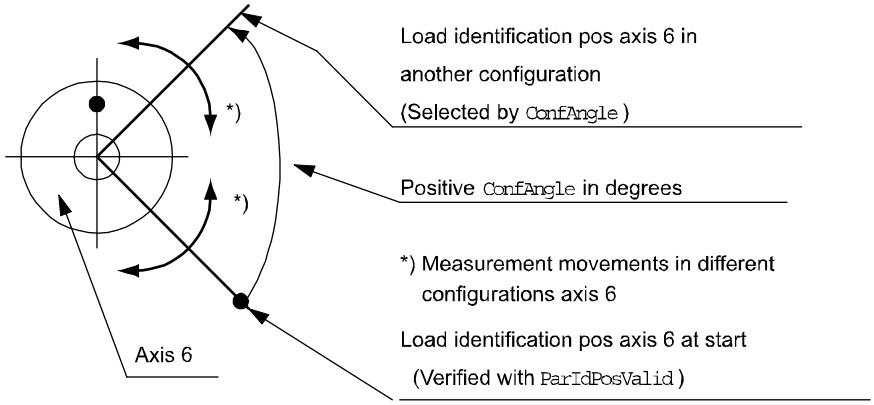
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[ \ConfAngle ]`

Datový typ: `num`

Volitelný argument pro určení konkrétního konfiguračního úhlu ± stupňů, který bude použit pro identifikaci parametru.



xx0500002493

Výchozí hodnota + 90 stupňů, jestliže tento argument není určen.

Min. + nebo - 30 stupňů. Optimum + nebo - 90 stupňů.

Řešení chyb

Jestliže se objeví chyba, systémová proměnná `ERRNO` je nastavena na `ERR_PID_RAISE_PP`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
ParIdPosValid '('
  [ ParIdType ':=' ] <expression (IN) of paridnum> ','
  [ Pos ':=' ] <variable (VAR) of jointtarget> ','
  [ AxValid ':=' ] <array variable {*} (VAR) of bool>
  [ '\' ConfAngle ':=' <expression (IN) of num> ] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Typ identifikace parametru	*paridnum - Typ identifikace parametru na str 1547*
Platný typ robotu	*ParIdRobValid - Platný typ robotu pro identifikaci parametru na str 1250*
Identifikace zatížení nástroje nebo užitečné zátěže	*LoadId - Identifikace zatížení nástroje nebo užitečné zátěže na str 332*
Identifikace zátěže polohovačů (IRBP).	*ManLoadIdProc - Identifikace zátěže IRBP manipulátorů na str 339*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.118 ParIdRobValid - Platný typ robotu pro identifikaci parametru

Použití

`ParIdRobValid` (*Parameter Identification Robot Valid*) kontroluje, jestli typ robotu nebo manipulátoru je platný pro identifikaci aktuálního parametru, jako je identifikace zátěže nástroje nebo užitečné zátěže.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému *MultiMove* v úlohách Motion.

Základní příklady

Následující příklad názorně ukazuje funkci`ParIdRobValid`.

Příklad 1

```
TEST ParIdRobValid (TOOL_LOAD_ID)
  CASE ROB_LOAD_VAL:
    ! Possible to do load identification of tool in actual robot
         type
    ...
  CASE ROB_LM1_LOAD_VAL:
    ! Only possible to do load identification of tool with
    ! IRB 6400FHD if actual load < 200 kg
    ...
  CASE ROB_NOT_LOAD_VAL:
    ! Not possible to do load identification of tool in actual
         robot type
    ...
ENDTEST
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `paridvalidnum`

Jestli může být provedena identifikace určeného parametru s aktuálním typem robotu nebo manipulátoru, jak je definováno v tabulce dole.

Hodnota	Symbolická konstanta	Komentář
10	ROB_LOAD_VAL	Platný typ robotu nebo manipulátoru pro identifikaci aktuálního parametru.
11	ROB_NOT_LOAD_VAL	Neplatný typ pro identifikaci aktuálního parametru
12	ROB_LM1_LOAD_VAL	Platný typ robotu IRB 6400FHD pro identifikaci aktuálního parametru, jestliže aktuální zátěž < 200 kg.

Argumenty

```
ParIdRobValid(ParIdType [\MechUnit] [\AxisNo])
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`ParIdType`

Datový typ: `paridnum`

Identifikace typu parametru, jak je definováno v tabulce dole.

Hodnota	Symbolická konstanta	Komentář
1	TOOL_LOAD_ID	Identifikovat zatížení nástroje robotu
2	PAY_LOAD_ID	Identifikovat užitečnou zátěž robotu (Ref. instrukce GripLoad)
3	IRBP_K	Identifikovat externí manipulátor IRBP K zátěž
4	IRBP_L	Identifikovat externí manipulátor IRBP L zátěž
4	IRBP_C	Identifikovat externí manipulátor IRBP C zátěž
4	IRBP_C_INDEX	Identifikovat externí manipulátor IRBP C_INDEX zátěž
4	IRBP_T	Identifikovat externí manipulátor IRBP T zátěž
5	IRBP_R	Identifikovat externí manipulátor IRBP R zátěž
6	IRBP_A	Identifikovat externí manipulátor IRBP A zátěž
6	IRBP_B	Identifikovat externí manipulátor IRBP B zátěž
6	IRBP_D	Identifikovat externí manipulátor IRBP D zátěž

`[ \MechUnit ]`

Mechanical Unit

Datový typ: `mecunit`

Mechanická jednotka použitá pro identifikaci zatížení. Bude určeno pouze pro externí manipulátor. Jestliže tento argument je vypuštěn, použije se TCP robot v úloze.

`[ \AxisNo ]`

Axis number

Datový typ: `num`

Číslo osy v rámci mechanické jednotky, která drží zátěž k identifikaci. Bude určeno pouze pro externí manipulátor.

Když je použit argument `\MechUnit`, potom musí být použit `\AxisNo`. Argument `\AxisNo` není možné používat bez `\MechUnit`.

Řešení chyb

Jestliže se objeví chyba, systémová proměnná `ERRNO` je nastavena na `ERR_PID_RAISE_PP`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
ParIdRobValid '('
  [ParIdType ':='] <expression (IN) of paridnum>
  ['\' MechUnit ':=' <variable (VAR) of mecunit>]
  ['\' AxisNo ':=' <expression (IN) of num>] ')'
```

Funkce s vrácenou hodnotou datového typu `paridvalidnum`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.118 ParIdRobValid - Platný typ robotu pro identifikaci parametru

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Typ identifikace parametru	*paridnum - Typ identifikace parametru na str 1547*
Mechanická jednotka, která bude identifikována	*mecunit - Mechanická jednotka na str 1531*
Výsledek této funkce	*paridvalidnum - Výsledek ParIdRobValid na str 1549*
Platná pozice robotu	*ParIdPosValid - Platná pozice robotu pro identifikaci parametru na str 1247*
Identifikace zatížení nástroje robotu nebo užitečné zátěže	*LoadId - Identifikace zatížení nástroje nebo užitečné zátěže na str 332*
Identifikace zátěže polohovačů	*ManLoadIdProc - Identifikace zátěže IRBP manipulátorů na str 339*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.119 PathLevel - Získat aktuální úroveň dráhy

Použití

`PathLevel` se používá k získání aktuální úrovně dráhy. Tato funkce ukáže, jestli je úloha vykonávána na původní úrovni nebo jestli původní dráha pohybu byla uložena a je vykonáván nový dočasný pohyb. Přečtěte si víc o *Path Recovery* v *Application manual - Controller software IRC5*.

Základní příklady

Následující příklad názorně ukazuje funkci`PathLevel`.

Viz také *Další příklady na str 1253*.

Příklad 1

```
VAR num level;
level:= PathLevel();
```

Proměnná `level` bude 1, jestliže je vykonávána v původní dráze pohybu, nebo 2, jestliže je vykonávána v nové, dočasné dráze pohybu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Existují dvě možné vrácené hodnoty.

Vratná hodnota (Vrátit hodnotu)	Popis
1	Vykonávání v původní dráze pohybu.
2	Vykonávání v úrovni StorePath, dočasně nové dráze pohybu.

Další příklady

Další příklad, jak používat funkci `PathLevel`, je názorně uveden dole.

Příklad 1

```
...
MoveL p100, v100, z10, tool1;
StopMove;
StorePath;
p:= CRobT(\Tool:=tool1);
!New temporary movement
MoveL p1, v100, fine, tool1;
...
level:= PathLevel();
...
MoveL p, v100, fine, tool1;
RestoPath;
StartMove;
...
```

Proměnná `level` bude 2.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.119 PathLevel - Získat aktuální úroveň dráhy

RobotWare - OS

Pokračování

Omezení

Doplněk RobotWare Path Recovery musí být nainstalován, aby bylo možné používat funkci `PathLevel` na úrovni dráhy 2.

Syntaxe

```
PathLevel '(' ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Obnova dráhy	*Application manual - Controller software IRC5*
Uložit a obnovit dráhu	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742* *RestoPath - Obnovuje cestu po přerušení na str 546*
Zastavit a spustit pohyb	*StartMove - Znovu spouští pohyb robotu na str 707* *StopMove - Zastavuje pohyby robotu na str 736*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.120 PathRecValidBwd - Je zaznamenána platná zpětná dráha

Použití

`PathRecValidBwd` se používá pro kontrolu, jestli záznamník dráhy je aktivní a jestli je dostupná zaznamenaná zpětná dráha.

Základní příklady

Následující příklad názorně ukazuje funkci`PathRecValidBwd`.

Viz také *Další příklady na str 1257*.

Příklad 1

```
VAR bool bwd_path;
VAR pathrecid fixture_id;
bwd_path := PathRecValidBwd (\ID:=fixture_id);
```

Proměnná `bwd_path` je nastavena na TRUE, jestliže je možné udělat zálohu k pozici s identifikátorem `fixture_id`. Jestliže nikoliv, `bwd_path` je nastaven na FALSE.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.120 PathRecValidBwd - Je zaznamenána platná zpětná dráha

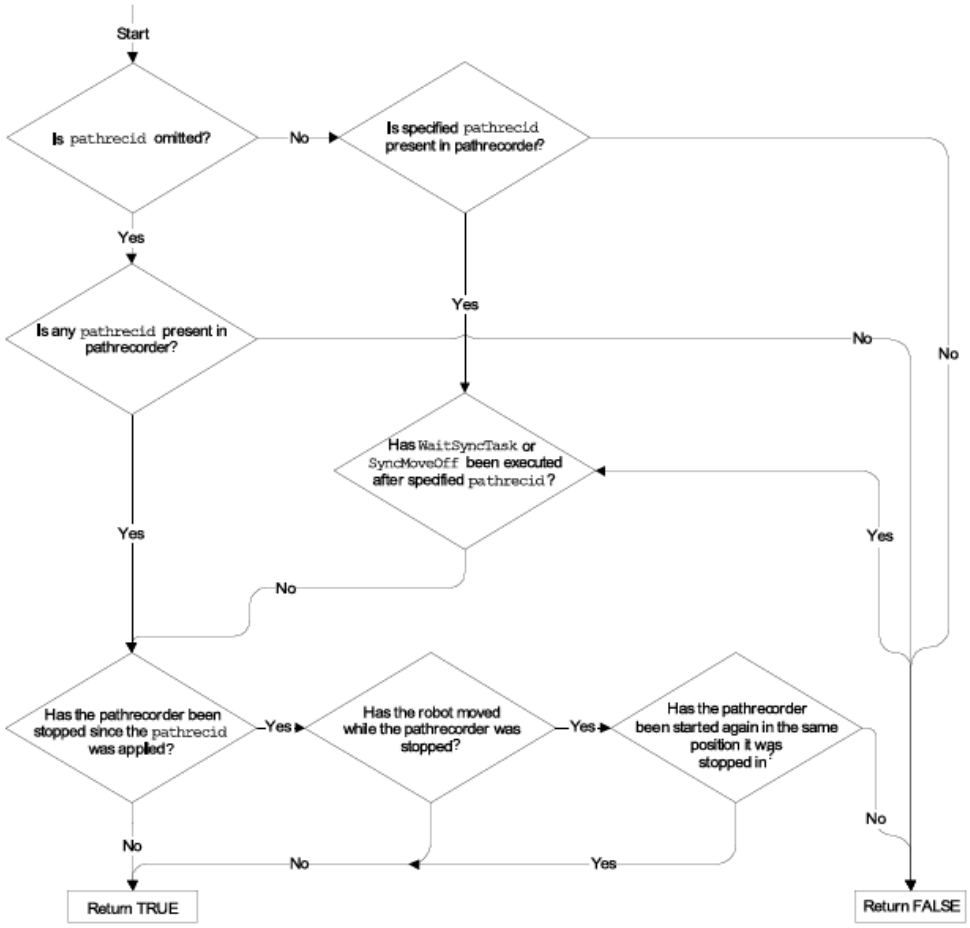
Path Recovery

Pokračování

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Vrácená hodnota funkce může být stanovena z následujícího vývojového diagramu:



xx0500002132

Argumenty

```
PathRecValidBwd ([\ID])
```

`[\ID]`

Identifier

Datový typ: `pathrecid`

Proměnná, která určuje jméno pozice začátku záznamu. Datový typ `pathrecid` je nehodnotový typ, použitý pouze jako identifikátor pro pojmenování záznamové pozice.

Vykonávání programu

Předtím, než je záznamník dráhy přikázán k pohybu dozadu s `PathRecMoveBwd`, je možné zkontrolovat, jestli je přítomna platná zaznamenaná dráha s `PathRecValidBwd`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Následující příklady názorně ukazují funkci`PathRecValidBwd`.

Příklad 1

```
PathRecStart id1;
MoveL p1, vmax, z50, tool1;
MoveL p2, vmax, z50, tool1;
bwd_path := PathRecValidBwd (\ID := id1);
```

Záznamník dráhy je spuštěn a dvě pohybové instrukce jsou vykonány. `PathRecValidBwd` vrátí TRUE a dostupná záložní dráha bude:

Od `p2` k `p1` k pozici startu.

Příklad 2

```
PathRecStart id1;
MoveL p1, vmax, z50, tool1;
MoveL p2, vmax, z50, tool1;
PathRecStop \Clear;
bwd_path:= PathRecValidBwd (\ID := id1);
```

Záznamník dráhy je spuštěn a dvě pohybové instrukce jsou vykonány. Potom je záznamník dráhy zastaven a vynulován. `PathRecValidBwd` vrátí FALSE.

Příklad 3

```
PathRecStart id1;
MoveL p1, vmax, z50, tool1;
PathRecStart id2;
MoveL p2, vmax, z50, tool1;
bwd_path := PathRecValidBwd ();
```

Záznamník dráhy je spuštěn a jedna pohybová instrukce je vykonána. Potom je spuštěn pomocný identifikátor dráhy, následovaný pohybovou instrukcí. `PathRecValidBwd` vrátí TRUE a záložní dráha bude:

Od `p2` k `p1`.

Příklad 4

```
PathRecStart id1;
MoveL p1, vmax, z50, tool1;
WaitSyncTask sync101, tasklist_r1o1;
MoveL p2, vmax, z50, tool1;
bwd_path1 := PathRecValidBwd ();
bwd_path2 := PathRecValidBwd (\ID := id1);
```

Vykonávání shora uvedeného programu povede k výsledku, že booleánské proměnné `bwd_path1` bude přiděleno TRUE, jelikož existuje platná zpětná dráha k `WaitSyncTask`. Booleánské proměnné `bwd_path2` bude přiděleno FALSE, jelikož není možné zálohovat nad příkazem `WaitSyncTask`.

Syntaxe

```
PathRecValidBwd '('
  ['\' ID ':=' < variable (VAR) of pathrecid >] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.120 PathRecValidBwd - Je zaznamenána platná zpětná dráha

Path Recovery

Pokračování

Související informace

Pro informace o	Viz
Identifikátory záznamníku dráhy	*pathrecid - Identifikátor záznamníku dráhy na str 1551*
Spustit - zastavit záznamník dráhy	*PathRecStart - Spustit záznamník dráhy na str 467* *PathRecStop - Zastavit záznamník dráhy na str 470*
Přehrát záznamník dráhy dozadu	*PathRecMoveBwd - Posunout záznamník dráhy dozadu na str 458*
Zkontrolujte, jestli existuje platná dráha dopředu	*PathRecValidFwd - Je zaznamenána platná dráha dopředu na str 1259*
Přehrát záznamník dráhy dopředu	*PathRecMoveFwd - Posunout záznamník dráhy dopředu na str 464*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.121 PathRecValidFwd - Je zaznamenána platná dráha dopředu

Použití

`PathRecValidFwd` se používá ke kontrole, jestli může být použit záznamník dráhy pro pohyb dopředu. Schopnost pohybu dopředu se záznamníkem dráhy naznačuje, že záznamník dráhy musel být již dříve přikázán k pohybu dozadu.

Základní příklady

Následující příklad názorně ukazuje funkci`PathRecValidFwd`.

Viz také *Další příklady na str 1260*.

Příklad 1

```
VAR bool fwd_path;
VAR pathrecid fixture_id;

fwd_path:= PathRecValidFwd (\ID:=fixture_id);
```

Proměnná `fwd_path` je nastavena na `TRUE`, jestliže je možné udělat pohyb dopředu k pozici s identifikátorem `fixture_id`. Jestliže nikoliv, `fwd_path` je nastavena na `FALSE`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Vrácená hodnota `PathRecValidFwd` **bez** určeného `\ID` je:

`TRUE`, jestliže:

- Záznamník dráhy posunul robot dozadu pomocí `PathRecMoveBwd`.
- Robot se neposunul pryč od dráhy provedené `PathRecMoveBwd`.

`FALSE`, jestliže:

- Shora uvedené podmínky nejsou splněny.

Vrácená hodnota `PathRecValidFwd` **s** určeným `\ID` je:

`TRUE`, jestliže:

- Záznamník dráhy posunul robot dozadu pomocí `PathRecMoveBwd`.
- Robot se neposunul pryč od dráhy provedené `PathRecMoveBwd`.
- Uvedené `\ID` bylo překročeno během pohybu zpět.

`FALSE`, jestliže:

- Shora uvedené podmínky nejsou splněny.

Argumenty

```
PathRecValidFwd ([\ID])
```

`[\ID]`

Identifier

Datový typ: `pathrecid`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.121 PathRecValidFwd - Je zaznamenána platná dráha dopředu

Path Recovery

Pokračování

Proměnná, která určuje jméno pozice začátku záznamu. Datový typ `pathrecid` je nehodnotový typ, použitý pouze jako identifikátor pro pojmenování záznamové pozice.

Vykonávání programu

Poté, co byl záznamník dráhy přikázán k pohybu dozadu pomocí `PathRecMoveBwd`, je možné kontrolovat, jestli platná zaznamenaná dráha pro pohyb robotu dopředu existuje. Jestliže identifikátor `\ID` je vynechán, `PathRevValidFwd` vrací, jestliže je možné pohybovat se dopředu k pozici, kde byl pohyb dozadu iniciován.

Další příklady

Následující příklad názorně ukazuje funkci`PathRecValidFwd`.

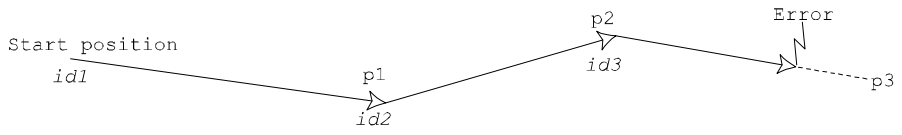
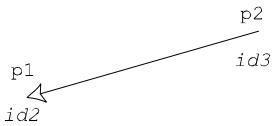
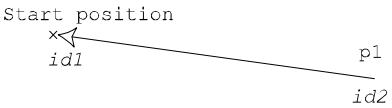
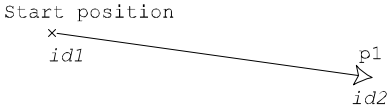
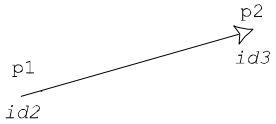
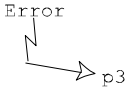
Příklad 1

```
VAR pathrecid id1;
VAR pathrecid id2;
VAR pathrecid id3;

PathRecStart id1;
MoveL p1, vmax, z50, tool1;
PathRecStart id2;
MoveL p2, vmax, z50, tool1;
PathRecStart id3;
!See figures 1 and 8 in tbe following table.
MoveL p3, vmax, z50, tool1;
ERROR
  StorePath;
  IF PathRecValidBwd(\ID:=id3) THEN
    !See figure 2 in the following table.
    PathRecMoveBwd \ID:=id3;
    ! Do some other operation
  ENDIF
  IF PathRecValidBwd(\ID:=id2) THEN
    !See figure 3 in the following table.
    PathRecMoveBwd \ID:=id2;
    ! Do some other operation
  ENDIF
  !See figure 4 in the following table.
  PathRecMoveBwd;
  ! Do final service action
  IF PathRecValidFwd(\ID:=id2) THEN
    !See figure 5 in the following table.
    PathRecMoveFwd \ID:=id2;
    ! Do some other operation
  ENDIF
  IF PathRecValidFwd(\ID:=id3) THEN
    !See figure 6 in the following table.
    PathRecMoveFwd \ID:=id3;
    ! Do some other operation
  ENDIF
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
!See figure 7 in the following table.
PathRecMoveFwd;
RestoPath;
StartMove;
RETRY;
```

1	 xx0500002121
2	 xx0500002124
3	 xx0500002126
4	 xx0500002127
5	 xx0500002128
6	 xx0500002129
7	 xx0500002130
8	 xx0500002131

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.121 PathRecValidFwd - Je zaznamenána platná dráha dopředu

Path Recovery
Pokračování

Příklad nahoře spustí záznamník dráhy a přidá identifikátory na tři různá místa podél vykonávané dráhy. Obrázek nahoře odkazuje na kód příkladu a popisuje, jak se robot bude pohybovat v případě chyby při vykonávání k bodu `p3`. Používají se `PathRecValidBwd` a `PathRecValidFwd` a není možné určit předem, kde v programu se případná chyba objeví.

Syntaxe

```
PathRecValidFwd '('
  ['\' ID ':=' < variable (VAR) of pathrecid >] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Identifikátory záznamníku dráhy	*pathrecid - Identifikátor záznamníku dráhy na str 1551*
Spustit - zastavit záznamník dráhy	*PathRecStart - Spustit záznamník dráhy na str 467* *PathRecStop - Zastavit záznamník dráhy na str 470*
Zkontrolujte, jestli existuje platná dráha dozadu	*PathRecValidBwd - Je zaznamenána platná zpětná dráha na str 1255*
Přehrát záznamník dráhy dozadu	*PathRecMoveBwd - Posunout záznamník dráhy dozadu na str 458*
Přehrát záznamník dráhy dopředu	*PathRecMoveFwd - Posunout záznamník dráhy dopředu na str 464*
Pohyb všeobecně	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.122 PFRestart - Zkontrolovat přerušenou dráhu po výpadku napájení

Použití

`PFRestart` *(Power Failure Restart)* se používá ke kontrole, jestli dráha byla přerušena při výpadku napájení. Jestliže ano, může být nutné provést některé konkrétní činnosti. Funkce kontroluje dráhu na aktuální úrovni, základnové úrovni a na úrovni přerušení.

Základní příklady

Následující příklad názorně ukazuje funkci`PFRestart`.

Příklad 1

```
IF PFRestart() = TRUE THEN
```

Je kontrolováno, jestli přerušená dráha existuje na aktuální úrovni. Jestliže ano, funkce vrátí `TRUE`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže přerušená dráha existuje na určené úrovni dráhy, jinak `FALSE`.

Argumenty

```
PFRestart([\Base] | [\Irpt])
```

`[ \Base ]`

Base Level

Datový typ: `switch`

Vrací `TRUE`, jestliže přerušená dráha existuje na základnové úrovni.

`[ \Irpt ]`

Interrupt Level

Datový typ: `switch`

Vrací `TRUE`, jestliže přerušená dráha existuje na úrovni `StorePath`.

Jestliže není dán žádný argument, funkce vrátí `TRUE`, jestliže přerušená dráha existuje na aktuální úrovni.

Syntaxe

```
PFRestart '('
  ['\' Base] | ['\' Irpt] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.123 PoseInv - Invertuje data pose

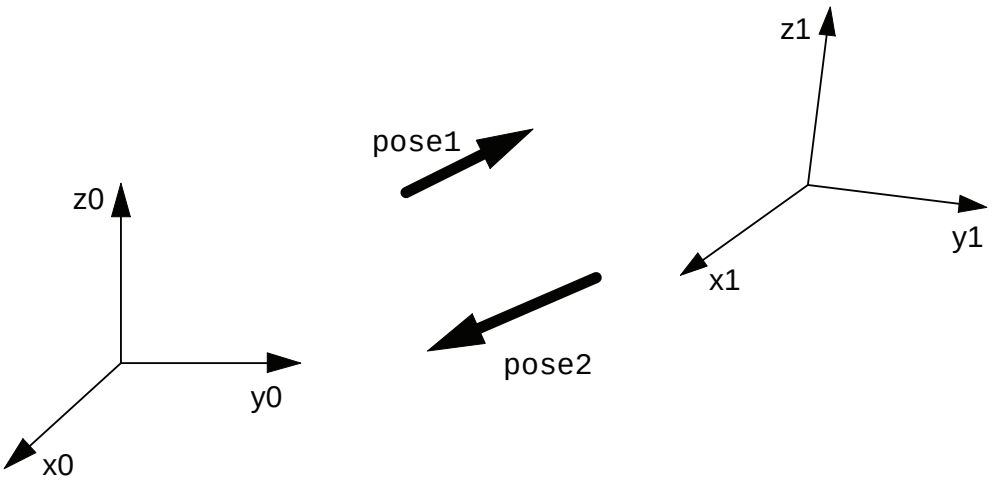
Použití

`PoseInv` (***Pose Invert***) vypočítává reverzní transformaci `pose`.

Základní příklady

Následující příklad názorně ukazuje funkci`PoseInv`.

Příklad 1



xx0500002443

`pose1` reprezentuje souřadný systém 1 ve vztahu k souřadnému systému 0. Transformace dávající souřadný systém 0 ve vztahu k souřadnému systému 1 je získána reverzní transformací uloženou v `pose2`.

```
VAR pose pose1;
VAR pose pose2;
...
pose2 := PoseInv(pose1);
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `pose`

Hodnota reverzní pose.

Argumenty

```
PoseInv (Pose)
```

`Pose`

Datový typ: `pose`

`pose` pro inverzi.

Syntaxe

```
PoseInv'('
  [Pose ':='] <expression (IN) of pose>
  ')'
```

Funkce s vrácenou hodnotou datového typu `pose`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.124 PoseMult - Násobí pose data

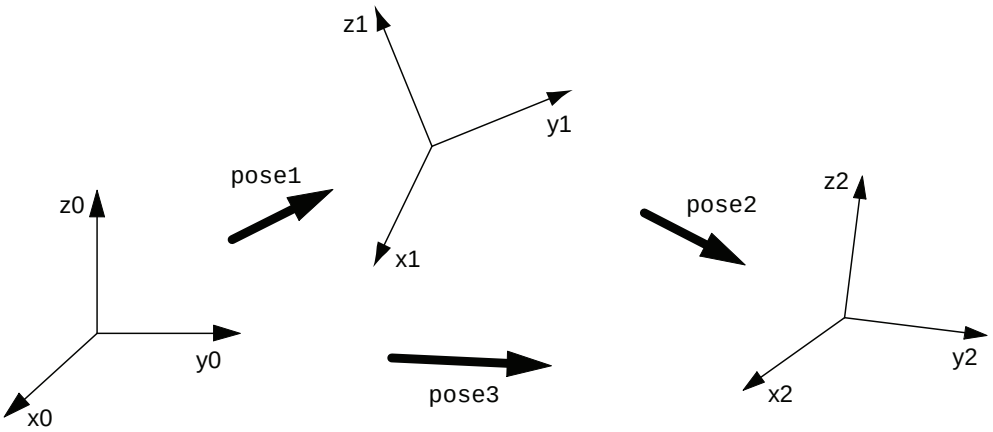
Použití

`PoseMult` (***Pose Multiply***) se používá k výpočtu produktu dvou pose transformací. Typické použití je při výpočtu nové pose jako výsledku posunu fungujícího na původní pose.

Základní příklady

Následující příklad názorně ukazuje funkci`PoseMult`.

Příklad 1



xx0500002444

`pose1` reprezentuje souřadný systém 1 ve vztahu k souřadnému systému 0. `pose2` reprezentuje souřadný systém 2 ve vztahu k souřadnému systému 1. Transformace dávající `pose3`, souřadný systém 2 ve vztahu k souřadnému systému 0, je získána produktem dvou transformací:

```
VAR pose pose1;
VAR pose pose2;
VAR pose pose3;
...
pose3 := PoseMult(pose1, pose2);
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `pose`

Hodnota produktu dvou pose.

Argumenty

```
PoseMult (Pose1 Pose2)
```

`Pose1`

Datový typ: `pose`

První pose.

`Pose2`

Datový typ: `pose`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Druhá pose.

Syntaxe

```
PoseMult '('
  [Pose1 ':='] <expression (IN) of pose> ','
  [Pose2 ':='] <expression (IN) of pose> ')'
```

Funkce s vrácenou hodnotou datového typu `pose`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.125 PoseVect - Aplikuje transformaci k vektoru

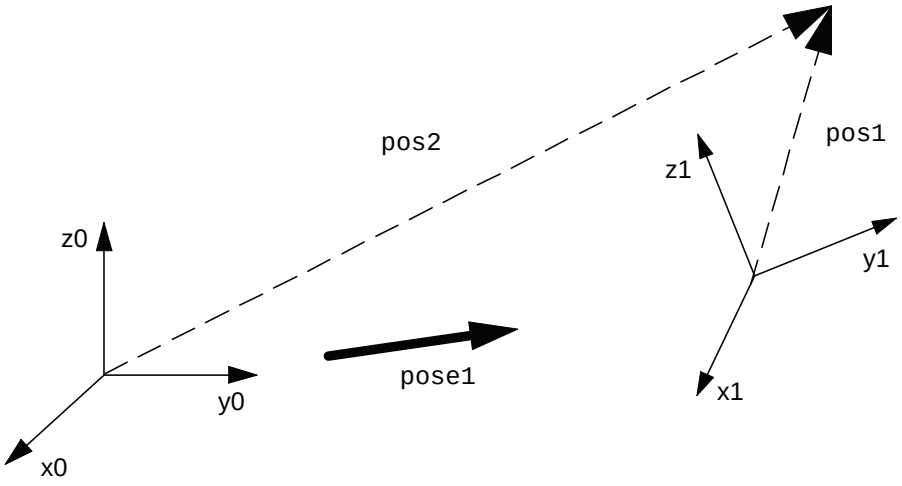
Použití

`PoseVect` (*Pose Vector*) se používá k výpočtu produktu pose a vektoru. Typicky se používá k výpočtu vektoru jako výsledku efektu posunu na původní vektor.

Základní příklady

Následující příklad názorně ukazuje funkci`PoseVect`.

Příklad 1



xx0500002445

`pose1` reprezentuje souřadný systém 1 ve vztahu k souřadnému systému 0.

`pos1` je vektor ve vztahu k souřadnému systému 1. Odpovídající vektor ve vztahu k souřadnému systému 0 je získán produktem;

```
VAR pose pose1;
VAR pos pos1;
VAR pos pos2;
...
...
pos2:= PoseVect(pose1, pos1);
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `pos`

Hodnota produktu pose a původní pos.

Argumenty

```
PoseVect (Pose Pos)
```

`Pose`

Datový typ: `pose`

Transformace, která bude použita.

`Pos`

Datový typ: `pos`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pos, která bude transformována.

Syntaxe

```
PoseVect '('
  [Pose ':='] <expression (IN) of pose> ','
  [Pos ':='] <expression (IN) of pos> ')'
```

Funkce s vrácenou hodnotou datového typu `pos`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.126 Pow - Vypočítává mocninu hodnoty

Použití

Pow (*Power*) se používá pro výpočet exponenciální hodnoty v jakékoliv základně

Základní příklady

Následující příklad názorně ukazuje funkci Pow.

Příklad 1

```
VAR num x;
VAR num y
VAR num reg1;
...
reg1:= Pow(x, y);
```

reg1 má přidělenu hodnotu x^y.

Vratná hodnota (Vrátit hodnotu)

Datový typ: num

Hodnota Base zvýšená na mocninu exponentu, tj. $^{\text{Exponentu}}$ základny.

Argumenty

```
Pow (Base Exponent)
```

Base

Datový typ: num

Hodnota argumentu základny.

Exponent

Datový typ: num

Hodnota argumentu exponentu.

Omezení

Vykonání funkce x^y vykáže chybu, jestliže:

- $x < 0$ a y není celé číslo;
- $x = 0$ a $y \leq 0$.

Syntaxe

```
Pow '('
  [Base ':='] <expression (IN) of num> ','
  [Exponent ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu num.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.127 PowDnum - Vypočítává mocninu hodnoty

Použití

`PowDnum` (*Power Dnum)* se používá pro výpočet exponenciální hodnoty v jakékoliv základně.

Základní příklady

Následující příklad názorně ukazuje funkci`PowDnum`.

Příklad 1

```
VAR dnum x;
VAR num y
VAR dnum value;
...
value:= PowDnum(x, y);
```

`value` má přidělenu hodnotu x^y.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Hodnota `Base` zvýšená na mocninu exponentu, tj. $^{\text{Exponentu}}$ základny.

Argumenty

```
PowDnum (Base Exponent)
```

`Base`

Datový typ: `dnum`

Hodnota argumentu základny.

`Exponent`

Datový typ: `num`

Hodnota argumentu exponentu.

Omezení

Vykonání funkce x^y vykáže chybu, jestliže:

- $x < 0$ a y není celé číslo;
- $x = 0$ a $y \leq 0$.

Syntaxe

```
PowDnum '('
  [Base ':='] <expression (IN) of dnum> ','
  [Exponent ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.127 PowDnum - Vypočítává mocninu hodnoty

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.128 PPMovedInManMode - Otestovat, jestli se ukazatel programu posunul v ručním režimu

Použití

`PPMovedInManMode` vrací TRUE, jestliže uživatel posunul ukazatel programu, zatímco řadič je v ručním režimu - to znamená, že klíč operátora je na Ruční omezené rychlosti nebo Ruční plné rychlosti. Posunutý stav ukazatele programu je resetován, když klíč je přepnut z Auto na Man nebo když se použije instrukce `ResetPPMoved`.

Základní příklady

Následující příklad názorně ukazuje funkci`PPMovedInManMode`.

Příklad 1

```
IF PPMovedInManMode() THEN
  WarnUserOfPPMovement;
  DoJob;
ELSE
  DoJob;
ENDIF
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

TRUE, jestliže ukazatel programu byl posunut uživatelem v ručním režimu.

Vykonávání programu

Otestovat, jestli ukazatel programu pro aktuální program byl posunut v ručním režimu.

Syntaxe

```
PPMovedInManMode '(' ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Otestovat, jestli ukazatel programu byl posunut	*IsStopStateEvent - Otestovat, jestli se ukazatel programu posunul na str 1216*
Resetovat stav posunutého ukazatele programu v ručním režimu	*ResetPPMoved - Resetovat stav pro ukazatel programu posunutý v ručním režimu na str 544*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.129 Present - Otestovat, jestli je použit volitelný parametr

Použití

`Present` se používá k otestování, jestli byl použit volitelný parametr při volání rutiny.

Volitelný parametr nesmí být použit, jestliže nebyl určen při volání rutiny. Tuto funkci je možné používat k testování, jestli byl parametr určen, aby se předešlo vzniku chyb.

Základní příklady

Následující příklad názorně ukazuje funkci`Present`.

Viz také *Další příklady na str 1274*.

Příklad 1

```
PROC feeder (\switch on | switch off)
  IF Present (on) Set do1;
  IF Present (off) Reset do1;
ENDPROC
```

Výstup `do1`, který kontroluje podavač, je nastaven nebo resetován podle argumentu použitého při volání rutiny.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

TRUE = Hodnota parametru nebo přepínač byly definovány při volání rutiny.

FALSE = Hodnota parametru nebo přepínač nebyly definovány.

Argumenty

```
Present (OptPar)
```

`OptPar`

Optional Parameter

Datový typ: Jakýkoliv typ

Jméno volitelného parametru, který má být otestován.

Další příklady

Následující příklad názorně ukazuje funkci`Present`.

Příklad 1

```
PROC glue (\switch on, num glueflow, robtarget topoint, speeddata
     speed, zonedata zone, PERS tooldata tool, \PERS wobjdata wobj)
  IF Present (on) PulseDO glue_on;
    SetAO gluesignal, glueflow;
  IF Present (wobj) THEN
    MoveL topoint, speed, zone, tool \WObj:=wobj;
  ELSE
    MoveL topoint, speed, zone, tool;
  ENDIF
ENDPROC
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Rutina lepení je provedena. Jestliže argument `\on` je určen při volání rutiny, je generován impuls na signálu `glue_on`. Robot potom nastavuje analogový výstup `gluesignal`, který kontroluje lepicí pistoli a posouvá se do koncové pozice. Jelikož parametr wobj volitelný, použijí se různé instrukce `MoveL` podle toho, jestli je tento argument použit nebo nikoliv.

Syntaxe

```
Present '('
  [OptPar ':='] <reference (REF) of any type> ')'
```

REF parametr vyžaduje, v tomto případě, volitelný parametr name.

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Parametry rutiny	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.130 ProgMemFree - Zjistit velikost volné paměti programu

Použití

`ProgMemFree` (*Program Memory Free* se používá ke zjištění velikosti volné paměti programu,

Základní příklady

Následující příklad názorně ukazuje funkci`ProgMemFree`.

Příklad 1

```
FUNC num module_size(string file_path)
  VAR num pgmfree_before;
  VAR num pgmfree_after;

  pgmfree_before:=ProgMemFree();
  Load \Dynamic, file_path;
  pgmfree_after:=ProgMemFree();
  Unload file_path;
  RETURN (pgmfree_before-pgmfree_after);
ENDFUNC
```

`ProgMemFree` se používá ve funkci, která vrací hodnotu množství paměti, kterou modul přiděluje do paměti programu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Velikost volné paměti programu v bajtech.

Syntaxe

```
ProgMemFree '(' ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Načíst programový modul	*Load - Načíst programový modul během provádění na str 328*
Zrušit načtení programového modulu	*UnLoad - Stáhnout programový modul během provádění na str 905*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.131 PrxGetMaxRecordpos - Získat max pozici senzoru

Použití

`PrxGetMaxRecordpos` se používá pro vrácení max pozice v mm aktivního záznamu.

Max pozice senzoru se může používat pro škálování nebo omezování argumentu `max_sync` v instrukci `SyncToSensor`.

Základní příklad

```
maxpos:=PrxGetMaxRecordpos Ssync1;
```

Získat max pozici pro aktivní profil u mechanické jednotky `Ssync1`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Max pozice (v mm) zaznamenaného profilu pohybu senzoru.

Argumenty

```
PrxGetMaxRecordpos MechUnit
```

`MechUnit`

Datový typ: `mechunit`

Pohybující se objekt mechanické jednotky, ke které je pohyb robotu synchronizován.

Vykonávání programu

Záznam musí být ukončen a záznam musí být aktivní.

Syntaxe

```
PrxGetMaxRecordpos '('
  [ MechUnit ':=' ] < expression (IN) of mechunit> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Machine Synchronization	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.132 RawBytesLen - Získat délku dat rawbytes

Použití

RawBytesLen se používá k získání aktuální délky platných bytů v proměnné rawbytes.

Základní příklady

Následující příklad názorně ukazuje funkci RawBytesLen.

Příklad 1

```
VAR rawbytes from_raw_data;
VAR rawbytes to_raw_data;
VAR num integer := 8
VAR num float := 13.4;
ClearRawBytes from_raw_data;
PackRawBytes integer, from_raw_data, 1 \IntX := INT;
PackRawBytes float, from_raw_data, (RawBytesLen(from_raw_data)+1)
     \Float4;
CopyRawBytes from_raw_data, 1, to_raw_data, 3;
```

V tomto příkladu je proměnná from_raw_data typu rawbytes nejprve vynulována, tj. všechny bajty nastaveny na 0 (stejné jako výchozí hodnota u deklarace). Potom je hodnota celého čísla umístěna do prvních 2 bajtů a s pomocí funkce RawBytesLen je hodnota float umístěna do dalších 4 bajtů (se začátkem na indexu 3).

Po naplnění from_raw_data daty je obsah (6 bajtů) kopírován do to_raw_data, se začátkem na pozici 3.

Vratná hodnota (Vrátit hodnotu)

Datový typ: num

Aktuální délka platných bajtů v proměnné typu rawbytes; rozsah 0 ... 1024.

Obecně, aktuální délka platných bajtů v proměnné rawbytes je aktualizována systémem, aby to byl poslední zapsaný bajt ve struktuře rawbytes.

Pokud jde o podrobnosti, viz datový typ rawbytes, instrukce ClearRawBytes, CopyRawBytes, PackDNHeader, PackRawBytes, a ReadRawBytes.

Argumenty

```
RawBytesLen (RawData)
```

RawData

Datový typ: rawbytes

RawData je datový kontejner, jehož aktuální délka platných bajtů bude vrácena.

Vykonávání programu

Během vykonávání programu je aktuální délka platných bajtů vrácena.

Syntaxe

```
RawBytesLen '('
[RawData ':='] < variable (VAR) of rawbytes> ')'
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Data`rawbytes`	*rawbytes - Data raw na str 1559*
Vyčistit obsah dat `rawbytes`	*ClearRawBytes - Vyčistit obsahy rawbyte dat na str 118*
Kopírovat obsah dat `rawbytes`	*CopyRawBytes - Kopírovat obsah dat rawbytes na str 137*
Zabalit hlavičku DeviceNet do dat `rawbytes`	*PackDNHeader - Zabalit hlavičku DeviceNet do dat rawbytes na str 446*
Zabalit data do dat `rawbytes`	*PackRawBytes - Zabalit data do dat rawbytes na str 449*
Načíst data `rawbytes`	*ReadRawBytes - Přečíst data rawbytes na str 530*
Rozbalit data z dat `rawbytes`	*UnpackRawBytes - Rozbalit data z rawbyte dat na str 908*
Zapsat data `rawbytes`	*WriteRawBytes - Zapsat data rawbytes na str 994*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.133 ReadBin - Přečte bajt ze souboru nebo sériového kanálu

Použití

`ReadBin` (*Read Binary*) se používá ke čtení bajtu (8 bitů) ze souboru nebo sériového kanálu.

Tato funkce funguje na souborech binárních i založených na znacích nebo na sériových kanálech.

Základní příklady

Následující příklad názorně ukazuje funkci`ReadBin`.

Viz také *Další příklady na str 1281*.

Příklad 1

```
VAR num character;
VAR iodev inchannel;
...
Open "com1:", inchannel\Bin;
character := ReadBin(inchannel);
```

Bajt je přečten z binárního sériového kanálu `inchannel`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Bajt (8 bitů) je čten z určeného souboru nebo sériového kanálu. Tento bajt je převeden na odpovídající kladnou numerickou hodnotu a vrácen jako datový typ `num`. Jestliže soubor je prázdný (konec souboru), je vráceno `EOF_BIN` (číslo -1).

Argumenty

```
ReadBin (IODevice [\Time])
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) souboru nebo sériového kanálu, který bude přečten.

`[\Time]`

Datový typ: `num`

Max. čas pro čtecí operaci (vypršení času) v sekundách. Jestliže tento argument není určen, potom je max čas nastaven na 60 sekund. Má-li se čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Jestliže tento čas uběhne před dokončením čtecí operace, potom bude volán chybový handler s chybovým kódem `ERR_DEV_MAXTIME`. Jestliže neexistuje chybový handler, vykonávání bude zastaveno.

Funkce vypršení času se také používá během zastavení programu a bude uvedena programem RAPID při spouštění programu.

Vykonávání programu

Vykonávání programu čeká, až bude možné přečíst bajt (8 bitů) ze souboru nebo sériového kanálu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Další příklady

Následující příklad názorně ukazuje funkci`ReadBin`.

Příklad 1

```
VAR num bindata;
VAR iodev file;

Open "HOME:/myfile.bin", file \Read \Bin;
bindata := ReadBin(file);
WHILE bindata <> EOF_BIN DO
  TPWrite ByteToStr(bindata\Char);
  bindata := ReadBin(file);
ENDWHILE
```

Čte obsah binárního souboru myfile.bin od začátku do konce a zobrazí přijatá binární data převedená na znaky na FlexPendantu (jeden znak na každé řádce).

Omezení

Funkci je možné používat pouze pro soubory a sériové kanály, které byly otevřeny s přístupem ke čtení (`\Read` pro soubory na základě znaků, `\Bin` nebo `\Append \Bin` pro binární soubory).

Řešení chyb

Jestliže se objeví chyba během čtení, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`.

Jestliže vyprší čas před dokončením čtecí operace, potom je systémová proměnná `ERRNO` nastavena na `ERR_DEV_MAXTIME`

Tyto chyby mohou být potom ošetřeny chybovým handlerem.

Předdefinovaná data

Konstanta `EOF_BIN` může být použita k zastavení čtení na konci souboru.

```
CONST num EOF_BIN := -1;
```

Syntaxe

```
ReadBin '('
  [IODevice ':='] <variable (VAR) of iodev>
  ['\' Time ':=' <expression (IN) of num>] ')'
```

Funkce s vrácenou hodnotou typu `num`.

Související informace

Pro informace o	Viz
Otevření atd. souborů nebo sériových kanálů	*Technická referenční příručka - Přehled RAPID*
Konvertovat bajt na řetězcová data	*ByteToStr - Převádí byte na řetězcová data na str 1077*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.133 ReadBin - Přečte bajt ze souboru nebo sériového kanálu

RobotWare - OS

Pokračování

Pro informace o	Viz
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.134 ReadDir - Přečíst další vstupní data v adresáři

Použití

`ReadDir` se používá k vyhledání jména dalšího souboru nebo podadresáře pod adresářem, který byl otevřen s instrukcí `OpenDir`.

Dokud funkce vrací `TRUE`, může tam být víc souborů nebo podadresářů k nalezení.

Základní příklady

Následující příklad názorně ukazuje funkci`ReadDir`.

Viz také *Další příklady na str 1284*.

Příklad 1

```
PROC lsdir(string dirname)
  VAR dir directory;
  VAR string filename;
  OpenDir directory, dirname;
  WHILE ReadDir(directory, filename) DO
    TPWrite filename;
  ENDWHILE
  CloseDir directory;
ENDPROC
```

Tento příklad vytiskne jména všech souborů nebo podadresářů pod určeným adresářem.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Funkce vrátí `TRUE`, jestliže nalezla jméno, jinak `FALSE`.

Argumenty

```
ReadDir (Dev FileName)
```

`Dev`

Datový typ: `dir`

Proměnná s referencí k adresáři dodanému instrukcí `OpenDir`.

`FileName`

Datový typ: `string`

Nalezený soubor nebo jméno podadresáře.

Vykonávání programu

Tato funkce vrací bool, který stanovuje, jestli hledání jména bylo úspěšné nebo nikoliv.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.134 ReadDir - Přečíst další vstupní data v adresáři

RobotWare - OS

Pokračování

Další příklady

Více příkladů funkce `ReadDir` je názorně uvedeno dole.

Příklad 1

Tento příklad implementuje generický přechod funkce struktury adresáře.

```
PROC searchdir(string dirname, string actionproc)
  VAR dir directory;
  VAR string filename;
  IF IsFile(dirname \Directory) THEN
    OpenDir directory, dirname;
    WHILE ReadDir(directory, filename) DO
        ! .. and . is the parent and resp. this directory
        IF filename <> ".." AND filename <> "." THEN
          searchdir dirname+"/"+filename, actionproc;
        ENDIF
    ENDWHILE
    CloseDir directory;
  ELSE
    %actionproc% dirname;
  ENDIF
ERROR
  RAISE;
ENDPROC

PROC listfile(string filename)
  TPWrite filename;
ENDPROC

PROC main()
  ! Execute the listfile routine for all files found under the
  ! tree in HOME:
  searchdir "HOME:","listfile";
ENDPROC
```

Tento program přenáší strukturu adresáře pod "`HOME`:“ a u každého nalezeného souboru volá proceduru `listfile`. `searchdir` je generická část, která neví nic o začátku hledání nebo kterou rutinu by bylo třeba volat u každého souboru. Používá `IsFile` ke kontrole, jestli našel podadresář nebo soubor a používá mechanismus opožděné vazby pro volání procedury určené v `actionproc` pro všechny nalezené soubory. Rutina `actionproc` by měla být procedurou s jedním parametrem typu `string`.

Řešení chyb

Jestliže adresář není otevřen (viz `OpenDir`), systémová proměnná `ERRNO` je nastavena na `ERR_FILEACC`. Tato chyba může být potom ošetřena v chybovém handleru.

Syntaxe

```
ReadDir '('
  [ Dev ':=' ] < variable (VAR) of dir> ','
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
[ FileName ':=' ] < var or pers (INOUT) of string> ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Adresář	*dir - Struktura adresáře souborů na str 1484*
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Otevřít adresář	*OpenDir - Otevřít adresář na str 444*
Zavřít adresář	*CloseDir - Zavřít adresář na str 125*
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.135 ReadMotor - Čte aktuální úhly motoru

Použití

`ReadMotor` se používá ke čtení aktuálních úhlů různých motorů robotu a externích os. Primární použití této funkce je v kalibrační proceduře robotu.

Základní příklady

Následující příklad názorně ukazuje funkci`ReadMotor`.

Viz také *Další příklady na str 1287*.

Příklad 1

```
VAR num motor_angle2;
motor_angle2 := ReadMotor(2);
```

Aktuální úhel motoru druhé osy robotu je uložen do `motor_angle2`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Aktuální úhel motoru v radiánech stanovené osy robotu nebo externích os.

Argumenty

```
ReadMotor [\MecUnit ] Axis
```

`MecUnit`

Mechanical Unit

Datový typ: `mecunit`

Jméno mechanické jednotky, pro kterou bude přečtena osa. Jestliže tento argument je vynechán, bude přečtena osa připojeného robotu.

`Axis`

Datový typ: `num`

Číslo osy, která bude přečtena (1 - 6).

Vykonávání programu

Vrácený úhel motoru reprezentuje aktuální pozici v radiánech pro motor bez jakéhokoliv kalibračního ofsetu. Hodnota nemá vztah k pevné pozici robotu, pouze k vnitřní nulové pozici rozkladače, tj. normálně je nulová pozice rozkladače, nejbližší ke kalibrační pozici (rozdíl mezi nulovou pozicí rozkladače a kalibrační pozicí), hodnotou ofsetu kalibrace. Hodnota reprezentuje plný pohyb každé osy, přestože to může být několik otáček.

Omezení

Je pouze možné přečíst aktuální úhly motoru u mechanických jednotek, které jsou kontrolovány z aktuální programové úlohy. U nepohybových úloh je možné přečíst úhly u mechanických jednotek kontrolovaných připojenou pohybovou úlohou.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Následující příklad názorně ukazuje funkci`ReadMotor`.

Příklad 1

```
VAR num motor_angle;
motor_angle := ReadMotor(\MecUnit:=STN1, 1);
```

Aktuální úhel motoru první osy `STN1` je uložen do `motor_angle`.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_AXIS_PAR`	Parametr osa v instrukci je nesprávný.

Syntaxe

```
ReadMotor '('
  ['\' MecUnit ':=' < variable (VAR) of mecunit> ',']
  [Axis ':=' ] < expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Čtení aktuálního úhlu spoje	*CJointT - Čte aktuální úhly spoje na str 1103*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.136 ReadNum - Přečte číslo ze souboru nebo sériového kanálu

Použití

`ReadNum` (*Read Numeric*) se používá ke čtení čísla ze souboru založeného na znacích nebo ze sériového kanálu.

Základní příklady

Následující příklad názorně ukazuje funkci`ReadNum`.

Viz také *Další příklady na str 1289*.

Příklad 1

```
VAR iodev infile;
...
Open "HOME:/file.doc", infile\Read;
reg1 := ReadNum(infile);
```

Pro `reg1` je přiděleno číslo přečtené ze souboru `file.doc`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Numerická hodnota přečtená z určeného souboru nebo sériového kanálu. Jestliže soubor je prázdný (konec souboru), je vráceno číslo větší než `EOF_NUM` (`9.998E36`).

Argumenty

```
ReadNum (IODevice [\Delim] [\Time])
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) souboru nebo sériového kanálu, který bude přečten.

`[\Delim]`

Delimiters

Datový typ: `string`

Řetězec obsahující oddělovače pro použití při parsování řádky v souboru nebo sériovém kanálu. Podle výchozího nastavení (bez `\Delim`) je soubor čten řádka po řádce a znak pro posuv řádku (\0A) je jediný oddělovač, který parsování bere v úvahu. Když je použit argument `\Delim`, bude brán v úvahu jakýkoliv znak v určeném řetězcovém argumentu kvůli určení významné části řádky.

Při použití argumentu `\Delim` kontrolní systém vždy přidá znaky posuvu kurzoru na začátek řádky (`\0D`) (`\0A`) k oddělovačům určeným uživatelem.

Kvůli určení ne-alfanumerických znaků použijte `\xx`, kde`xx` je šestnáctková reprezentace kódu ASCII znaku (příklad: TAB je určen `\09`).

`[\Time]`

Datový typ: `num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Max. čas pro čtecí operaci (vypršení času) v sekundách. Jestliže tento argument není určen, potom je max čas nastaven na 60 sekund. Má-li se čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Jestliže tento čas uběhne před dokončením čtecí operace, potom bude volán chybový handler s chybovým kódem `ERR_DEV_MAXTIME`. Jestliže neexistuje chybový handler, vykonávání bude zastaveno.

Funkce vypršení času se také používá během zastavení programu a bude uvedena programem RAPID při spouštění programu.

Vykonávání programu

Funkce začíná na aktuální pozici souboru, čte a ruší všechny oddělovače záhlaví. Oddělovač záhlaví bez argumentu `\Delim` je znakem posuvu řádky. Oddělovače záhlaví s argumentem `\Delim` jsou jakékoliv znaky určené v argumentu `\Delim` plus znaky přechodu kurzoru na další řádku a posuvu řádky. Dále čte vše včetně dalšího znaku oddělovače (bude zrušen), ale nikoliv víc než 80 znaků. Jestliže významná část překračuje 80 znaků, zbytek znaků bude přečten při příštím čtení.

Řetězec, který je čten, je potom převeden na numerickou hodnotu; například "234.4" je převedeno na numerickou hodnotu 234.4.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Další příklady

Následující příklad názorně ukazuje funkci`ReadNum`.

Příklad 1

```
reg1 := ReadNum(infile\Delim:="\09");
IF reg1 > EOF_NUM THEN
  TPWrite "The file is empty";
  ...
```

Čte číslo v řádce, kde čísla jsou oddělena znaky TAB (`"\09"`) nebo SPACE (`" "`). Před použitím čísla přečteného ze souboru je provedena kontrola, aby bylo jisté, že soubor není prázdný.

POZNÁMKA

Použijte < nebo > (menší než nebo větší než) při kontrole, jestli soubor je prázdný. Nepoužívejte = (rovný s).

Omezení

Tuto instrukci je možné používat pouze u souborů založených na znacích, které byly otevřeny ke čtení.

Řešení chyb

Jestliže se objeví chyba přístupu během čtení, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`.

Jestliže došlo k pokusu o čtení nenumerických dat, systémová proměnná `ERRNO` je nastavena na `ERR_RCVDATA`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.136 ReadNum - Přečte číslo ze souboru nebo sériového kanálu

RobotWare - OS

Pokračování

Jestliže vyprší čas před dokončením čtecí operace, potom je systémová proměnná ERRNO nastavena na ERR_DEV_MAXTIME

Tyto chyby mohou být potom ošetřeny chybovým handlerem.

Předdefinovaná data

Konstanta EOF_NUM může být použita k zastavení čtení na konci souboru.

```
CONST num EOF_NUM := 9.998E36;
```

Syntaxe

```
ReadNum '('
  [IODevice ':='] <variable (VAR) of iodev>
  ['\' Delim ':=' <expression (IN) of string>]
  ['\' Time ':=' <expression (IN) of num>] ')'
```

Funkce s vrácenou hodnotou typu num.

Související informace

Pro informace o	Viz
Otevření atd. souborů nebo sériových kanálů	*Technická referenční příručka - Přehled RAPID*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.137 ReadStr - Přečte řetězec ze souboru nebo sériového kanálu

Použití

`ReadStr` (*Read String*) se používá ke čtení řetězce ze souboru založeného na znacích nebo ze sériového kanálu.

Základní příklady

Následující příklad názorně ukazuje funkci`ReadStr`.

Viz také *Další příklady na str 1292*.

Příklad 1

```
VAR string text;
VAR iodev infile;
...
Open "HOME:/file.doc", infile\Read;
text := ReadStr(infile);
```

Pro `text` je přidělen řetězec přečtený ze souboru `file.doc`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Řetězec přečtený z určeného souboru nebo sériového kanálu. Jestliže soubor je prázdný (konec souboru), je vrácen řetězec`"EOF"`.

Argumenty

```
ReadStr (IODevice [\Delim] [\RemoveCR] [\DiscardHeaders] [\Time])
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) souboru nebo sériového kanálu, který bude přečten.

`[\Delim]`

Delimiters

Datový typ: `string`

Řetězec obsahující oddělovače pro použití při parsování řádky v souboru nebo sériovém kanálu. Podle výchozího nastavení je soubor čten řádka po řádce a znak pro posuv řádku (\0A) je jediný oddělovač, který parsování bere v úvahu. Když je použit argument `\Delim`, bude brán v úvahu jakýkoliv znak v určeném řetězcovém argumentu kvůli určení významné části řádky.

Kvůli určení ne-alfanumerických znaků použijte `\xx`, kde`xx` je šestnáctková reprezentace kódu ASCII znaku (příklad: TAB je určen `09`).

`[\RemoveCR]`

Datový typ: `switch`

Přepínač použitý k odstranění znaku nové řádky na konci řádky při čtení PC souborů. V PC souborech je nová řádka určena návratem vozíku a posuvem řádky (CRLF). Při čtení řádky v takových souborech je znak nové řádky načten standardně

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

do vráceného řetězce. Při používání tohoto argumentu bude znak nové řádky přečten ze souboru, ale nikoliv vložen do vráceného řetězce.

`[\DiscardHeaders]`

Datový typ: `switch`

Tento argument určuje, jestli oddělovače záhlaví (určené v `\Delim` plus standardní posuv řádky) budou přeskočeny nebo nikoliv před přenosem dat do vraceného řetězce. Podle výchozího nastavení, jestliže první znak na aktuální pozici souboru je oddělovač, je přečten, ale nikoliv přenesen do vraceného řetězce, parsování řádky je zastaveno a vrácen bude prázdný řetězec. Jestliže je použit tento argument, všechny oddělovače zahrnuté do řádky bude přečteny ze souboru, ale zrušeny, a nebude provedeno žádné vrácení, dokud vracený řetězec nebude obsahovat data začínající u prvního neoddělovacího znaku v řádce.

`[\Time]`

Datový typ: `num`

Max. čas pro čtecí operaci (vypršení času) v sekundách. Jestliže tento argument není určen, potom je max čas nastaven na 60 sekund. Má-li se čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Jestliže tento čas uběhne před dokončením čtecí operace, potom bude volán chybový handler s chybovým kódem `ERR_DEV_MAXTIME`. Jestliže neexistuje chybový handler, vykonávání bude zastaveno.

Funkce vypršení času se také používá během zastavení programu a bude uvedena programem RAPID při spouštění programu.

Vykonávání programu

Se začátkem na aktuální pozici souboru, jestliže je použit argument `\DiscardHeaders`, funkce čte a ruší všechny oddělovače záhlaví (znaky posuvu řádky a každý znak určený v argumentu `\Delim`). Ve všech případech potom čte všechno až k dalšímu znaku oddělovače, ale ne více než 80 znaků. Jestliže významná část překračuje 80 znaků, zbytek znaků bude přečten při příštím čtení. Oddělovač, který způsobil zastavení parsování, je přečten ze souboru, ale nikoliv přenesen do vraceného řetězce. Jestliže posledním znakem v řetězci je znak nové řádky a je použit argument `\RemoveCR`, tento znak bude z řetězce odstraněn.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Další příklady

Následující příklady názorně ukazují funkci`ReadStr`.

Příklad 1

```
text := ReadStr(infile);
IF text = EOF THEN
  TPWrite "The file is empty";
  ...
```

Před použitím řetězce přečteného ze souboru je provedena kontrola kvůli ujištění, že soubor není prázdný.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Příklad 2

Zvažte soubor obsahující:

```
<LF><SPACE><TAB>Hello<SPACE><SPACE>World<CR><LF>
text := ReadStr(infile);
```

`text` **bude prázdný řetězec: první znak v souboru je standardní oddělovač** `<LF>`**.**

```
text := ReadStr(infile\DiscardHeaders);
```

text bude obsahovat `<SPACE><TAB>Hello<SPACE><SPACE>World<CR>`**: první znak v souboru, standardní oddělovač** `<LF>`**, je zrušen.**

```
text := ReadStr(infile\RemoveCR\DiscardHeaders);
```

`text` **bude obsahovat** `<SPACE><TAB>Hello<SPACE><SPACE>World:` **první znak v souboru, standardní oddělovač <LF>, je zrušen; koncový znak nové řádky je odstraněn**

```
text := ReadStr(infile\Delim:=" \09"\RemoveCR\DiscardHeaders);
```

text bude obsahovat `"Hello"`**: první znaky v souboru, které odpovídají buď standardními oddělovači** `<LF>` **nebo znakové sadě definované od** `\Delim` **(mezera a tab) jsou zrušeny. Data jsou potom přenesena až k prvnímu oddělovači, který je přečten ze souboru, ale nikoliv přenesena do řetězce. Nové vyvolání stejného příkazu vrátí** `"World"`**.**

Příklad 3

Zvažte soubor obsahující:

```
<CR><LF>Hello<CR><LF>
text := ReadStr(infile);
```

`text` **bude obsahovat** `<CR>` **(znak** `\0d`**): znaky** `<CR>` **a** `<LF>` **jsou přečteny ze souboru, ale pouze** `<CR>` **je přenesen do řetězce. Nové vyvolání stejného příkazu vrátí** `"Hello\0d"`**.**

```
text := ReadStr(infile\RemoveCR);
```

`text` **bude obsahovat prázdný řetězec: znaky** `<CR>` **a** `<LF>` **jsou přečteny ze souboru;** `<CR>` **je přenesen, ale odstraněn z řetězce. Nové vyvolání stejného příkazu vrátí** `"Hello"`**.**

```
text := ReadStr(infile\Delim:="\0d");
```

`text` **bude obsahovat prázdný řetězec:** `<CR>` **je přečten ze souboru, ale nikoliv přenesen do vraceného řetězce. Nové vyvolání stejné instrukce vrátí znovu prázdný řetězec:** `<LF>` **je přečten ze souboru, ale nikoliv přenesen do vraceného řetězce.**

```
text := ReadStr(infile\Delim:="\0d"\DiscardHeaders);
```

`text` **bude obsahovat** `"Hello"`**. Nové vyvolání stejné instrukce vrátí** `"EOF"` **(konec souboru).**

Omezení

Funkci je možné používat pouze u sériových kanálů nebo souborů, které byly otevřeny pro čtení v režimu založeném na znacích.

Řešení chyb

Jestliže se objeví chyba během čtení, potom je systémová proměnná `ERRNO` **nastavena na** `ERR_FILEACC`**.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.137 ReadStr - Přečte řetězec ze souboru nebo sériového kanálu

RobotWare - OS

Pokračování

Jestliže vyprší čas před dokončením čtecí operace, potom je systémová proměnná ERRNO nastavena na ERR_DEV_MAXTIME

Tyto chyby mohou být potom ošetřeny chybovým handlerem.

Předdefinovaná data

Konstantu EOF je možné používat ke kontrole, jestli soubor byl prázdný, když došlo k pokusu čtení ze souboru nebo k zastavení čtení na konci souboru.

```
CONST string EOF := "EOF";
```

Syntaxe

```
ReadStr '('
[IODevice ':='] <variable (VAR) of iodev>
['\' Delim ':='<expression (IN) of string>]
['\' RemoveCR]
['\' DiscardHeaders]
['\' Time ':=' <expression (IN) of num>] ')'
```

Funkce s vrácenou hodnotou typu string.

Související informace

Pro informace o	Viz
Otevření atd. souborů nebo sériových kanálů	*Technická referenční příručka - Přehled RAPID*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.138 ReadStrBin - Čte řetězec z binárního sériového kanálu nebo souboru

Použití

`ReadStrBin` (*Read String Binary*) se používá pro čtení řetězce z binárního sériového kanálu nebo souboru.

Základní příklady

Následující příklad názorně ukazuje funkci`ReadStrBin`.

Příklad 1

```
VAR iodev channel;
VAR string text;
...
Open "com1:", channel \Bin;
text := ReadStrBin (channel, 10);
text := ReadStrBin(infile,20);
IF text = EOF THEN
```

Pro `text` je přidělen textový řetězec s `10` znaky načtený ze sériového kanálu odkazovaného od `channel`

Před použitím řetězce přečteného ze souboru je provedena kontrola kvůli ujištění, že soubor není prázdný.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Textový řetězec přečtený z určeného souboru nebo sériového kanálu. Jestliže soubor je prázdný (konec souboru), je vrácen řetězec`"EOF"`.

Argumenty

```
ReadStrBin (IODevice NoOfChars [\Time])
```

`IODevice`

Datový typ: `iodev`

Jméno (reference) binárního sériového kanálu nebo souboru, který bude načten.

`NoOfChars`

Number of Characters

Datový typ: `num`

Počet znaků, které budou přečteny z binárního sériového kanálu nebo souboru.

`[\Time]`

Datový typ: `num`

Max. čas pro čtecí operaci (vypršení času) v sekundách. Jestliže tento argument není určen, potom je max čas nastaven na 60 sekund. Má-li se čekat stále, použijte předdefinovanou konstantu `WAIT_MAX`.

Jestliže tento čas uběhne před dokončením čtecí operace, potom bude volán chybový handler s chybovým kódem `ERR_DEV_MAXTIME`. Jestliže neexistuje chybový handler, vykonávání bude zastaveno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.138 ReadStrBin - Čte řetězec z binárního sériového kanálu nebo souboru

RobotWare - OS

Pokračování

Funkce vypršení času se také používá během zastavení programu a bude uvedena programem RAPID při spouštění programu.

Vykonávání programu

Funkce čte určený počet znaků z binárního sériového kanálu nebo souboru.

Při restartu po selhání napájení bude každý otevřený soubor nebo sériový kanál v systému zavřen a I/O popisovač v proměnné typu `iodev` bude resetován.

Omezení

Tuto instrukci je možné používat pouze u sériových kanálů nebo souborů, které byly otevřeny pro čtení v binárním režimu.

Řešení chyb

Jestliže se objeví chyba během čtení, potom je systémová proměnná `ERRNO` nastavena na `ERR_FILEACC`.

Jestliže vyprší čas před dokončením čtecí operace, potom je systémová proměnná `ERRNO` nastavena na `ERR_DEV_MAXTIME`

Tyto chyby mohou být potom ošetřeny chybovým handlerem.

Předdefinovaná data

Konstantu `EOF` je možné používat ke kontrole, jestli soubor byl prázdný, když došlo k pokusu čtení ze souboru nebo k zastavení čtení na konci souboru.

```
CONST string EOF := "EOF";
```

Syntaxe

```
ReadStrBin '('
  [IODevice ':='] <variable (VAR) of iodev> ','
  [NoOfChars ':='] <expression (IN) of num>
  ['\' Time ':=' <expression (IN) of num>] ')'
```

Funkce s vrácenou hodnotou typu `string`.

Související informace

Pro informace o	Viz
Otevření atd. sériových kanálů nebo souborů	*Technická referenční příručka - Přehled RAPID*
Zapsat binární řetězec	*WriteStrBin - Zapisuje řetězec do binárního sériového kanálu na str 996*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.139 ReadVar - Přečíst proměnnou ze zařízení

Použití

`ReadVar` se používá pro čtení proměnné ze zařízení připojeného k sériovému rozhraní senzoru.

Rozhraní senzoru komunikuje se senzory přes sériové kanály pomocí transportního protokolu RTP1.

Toto je příklad konfigurace kanálu senzoru.

COM_PHY_CHANNEL:

- Name “COM1:”
- Connector “COM1”
- Baudrate 19200

COM_TRP:

- Name “sen1:”
- Type “RTP1”
- PhyChannel “COM1”

Základní příklady

Následující příklad názorně ukazuje funkci`ReadVar`.

Příklad 1

```
CONST num XCoord := 8;
CONST num YCoord := 9;
CONST num ZCoord := 10;

VAR pos SensorPos;

! Connect to the sensor device "sen1:" (defined in sio.cfg)
SenDevice "sen1:";

! Read a cartesian position from the sensor.

SensorPos.x := ReadVar ("sen1:", XCoord);
SensorPos.y := ReadVar ("sen1:", YCoord);
SensorPos.z := ReadVar ("sen1:", ZCoord);
```

Argumenty

```
ReadVar (device, VarNo, [ \TaskName ])
```

`device`

Datový typ: `string`

Jméno I/O zařízení konfigurované v sio.cfg pro použitý senzor.

`VarNo`

Datový typ: `num`

Argument `VarNo` se používá pro výběr proměnné ke čtení.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.139 ReadVar - Přečíst proměnnou ze zařízení

Sensor Interface

Pokračování

```
[ \TaskName ]
```

Datový typ: `string`

Argument `TaskName` umožňuje přístup k zařízením v jiných úkolech RAPID.

Řešení chyb

Chybová konstanta (hodnota ERRNO)	Popis
SEN_NO_MEAS	Selhání měření
SEN_NOREADY	Senzor není schopen zpracovat příkaz
SEN_GENERRO	Obecná chyba senzoru
SEN_BUSY	Snímač je zaneprázdněn
SEN_UNKNOWN	Neznámý senzor
SEN_EXALARM	Externí chyba senzoru
SEN_CAALARM	Interní chyba senzoru
SEN_TEMP	Chyba teploty senzoru
SEN_VALUE	Neplatná hodnota komunikace
SEN_CAMCHECK	Selhání kontroly senzoru
SEN_TIMEOUT	Chyba komunikace

Syntaxe

```
ReadVar
  [ device ':=' ] < expression(IN) of string> ','
  [ VarNo ':=' ] < expression (IN) of num > ','
  [ '\' TaskName ':=' < expression (IN) of string > ] ';'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Připojit k zařízení senzoru	*SenDevice - Připojit k zařízení senzoru na str 614*
Zapsat proměnnou senzoru	*WriteVar - Zapsat proměnnou na str 998*
Zapsat datový blok senzoru	*WriteBlock - Zapsat blok dat do zařízení na str 988*
Přečíst datový blok senzoru	*ReadBlock - přečíst blok dat ze zařízení na str 521*
Konfigurace komunikace senzoru	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.140 RelTool - Provést posun spojený s nástrojem

Použití

`RelTool` (*Relative Tool*) se používá s přidání posunu a/nebo rotace, vyjádřené v souřadném systému aktivního nástroje, k pozici robotu.

Základní příklady

Následující příklady názorně ukazují funkci`RelTool`.

Příklad 1

```
MoveL RelTool (p1, 0, 0, 100), v100, fine, tool1;
```

Robot je posunut na pozici, která je `100` mm od `p1` v z-směru nástroje.

Příklad 2

```
MoveL RelTool (p1, 0, 0, 0 \Rz:= 25), v100, fine, tool1;
```

Nástroj je otočen 25° kolem své osy z.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `robtarget`

Nová pozice s přídavkem posunu a/nebo rotace, pokud nějaké, ve spojení s aktivním nástrojem.

Argumenty

```
RelTool (Point Dx Dy Dz [\Rx] [\Ry] [\Rz])
```

`Point`

Datový typ: `robtarget`

Vstupní pozice robotu. Orientační část této pozice definuje aktuální orientaci souřadného systému nástroje.

`Dx`

Datový typ: `num`

Posunutí v mm ve směru x souřadného systému nástroje.

`Dy`

Datový typ: `num`

Posunutí v mm ve směru y souřadného systému nástroje.

`Dz`

Datový typ: `num`

Posunutí v mm ve směru z souřadného systému nástroje.

`[\Rx]`

Datový typ: `num`

Rotace ve stupních kolem osy x souřadného systému nástroje.

`[\Ry]`

Datový typ: `num`

Rotace ve stupních kolem osy y souřadného systému nástroje.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.140 RelTool - Provést posun spojený s nástrojem

RobotWare - OS

Pokračování

`[\Rz]`

Datový typ: `num`

Rotace ve stupních kolem osy z souřadného systému nástroje.

Jestliže jsou určeny dvě nebo tři rotace ve stejnou dobu, budou provedeny nejprve kolem osy x, potom kolem nové osy y a potom kolem nové osy z.

Syntaxe

```
RelTool '('
  [ Point ':=' ] < expression (IN) of robtarget> ','
  [Dx ':='] <expression (IN) of num> ','
  [Dy ':='] <expression (IN) of num> ','
  [Dz ':='] <expression (IN) of num>
  ['\' Rx ':=' <expression (IN) of num> ]
  ['\' Ry ':=' <expression (IN) of num> ]
  ['\' Rz ':=' <expression (IN) of num> ] ')'
```

Funkce s vrácenou hodnotou datového typu `robtarget`.

Související informace

Pro informace o	Viz
Poziční data	*robtarget - Poziční data na str 1572*
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.141 RemainingRetries - Zbývající nové pokusy, které se mají udělat

Použití

`RemainingRetries` se používá ke zjištění, kolik `RETRY` je třeba ještě udělat z chybového handleru v programu. Max počet nových pokusů je definován v konfiguraci.

Základní příklady

Následující příklad názorně ukazuje funkci`RemainingRetries`.

Příklad 1

```
...
ERROR
  IF RemainingRetries() > 0 THEN
    RETRY;
  ELSE
    TRYNEXT;
  ENDIF
...
```

Tento program znovu zkusí instrukci navzdory chybě, dokud nebude proveden max počet nových pokusů a potom zkusí další instrukci.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Vrácená hodnota ukazuje, kolik z max počtu nových pokusů ještě zbývá udělat.

Syntaxe

```
RemainingRetries '(' ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Obslužné programy pro řešení chyb	*Technická referenční příručka - Přehled RAPID*
Obnovit vykonávání po chybě	*RETRY - Obnovit vykonávání po chybě na str 548*
Konfigurovat max počet nových pokusů	*Technická referenční příručka - Systémové parametry*, sekce *System misc*.
Resetovat počet spočítaných nových pokusů	*ResetRetryCount - Resetovat počet pokusů na str 545*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.142 RMQGetSlotName - Získat jméno klienta RMQ

Použití

`RMQGetSlotName` (*RAPID Mesasage Queue Get Slot Name*) se používá k získání jména slotu RMQ nebo SDK klienta z identity daného slotu - tj. z daného `rmqslot`.

Základní příklady

Následující příklad názorně ukazuje funkci`RMQGetSlotName`.

Příklad 1

```
VAR rmqslot slot;
VAR string client_name;
RMQFindSlot slot, "RMQ_T_ROB1";
...
client_name := RMQGetSlotName(slot);
TPWrite "Name of the client: " + client_name;
```

Příklad ukazuje, jak získat jméno klienta pomocí identity klienta.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Jméno klienta je vráceno. Může to být jméno RMQ nebo jméno klienta Robot Application Builder pomocí funkčnosti RMQ.

Argumenty

```
RMQGetSlotName (Slot)
```

`Slot`

Datový typ: `rmqslot`

Číslo slotu identity klienta ke hledání jména.

Vykonávání programu

Instrukce `RMQGetSlotName` se používá k vyhledání jména klienta s určeným číslem identity stanoveným v argumentu `Slot`. Klient může být další RMQ nebo SDK klient.

Řešení chyb

Následující odstranitelné chyby mohou být generovány. Chyby je možné řešit v chybovém handleru `ERROR`. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_RMQ_INVALID`	Cílový slot nebyl připojen nebo cílový slot už není dostupný. Jestliže není připojen, musí být provedeno volání k `RMQFindSlot`. Jestliže není dostupný, důvodem je odpojení vzdáleného klienta od řadiče.

Syntaxe

```
RMQGetSlotName '('
  [ Slot ':=' ] < variable (VAR) of rmqslot > ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Popis funkčnosti RAPID Message Queue	*Application manual - Controller software IRC5*, sekce *RAPID Message Queue*.
Najděte číslo identity úlohy RAPID Message Queue nebo SDK klienta	*RMQFindSlot - Najít identitu slotu od jména slotu na str 554*
Odeslat data do fronty úlohy RAPID nebo SDK klienta	*RMQSendMessage - Odeslat datovou zprávu RMQ na str 568*
Získat první zprávu z fronty zpráv RAPID Message Queue.	*RMQGetMessage - Získat zprávu RMQ na str 556*
Odeslat data do fronty úlohy RAPID nebo SDK klienta a čekat na odpověď od klienta.	*RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu na str 572*
Vyjímá hlavičku dat z `rmqmessage`	*RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ na str 562*
Vyjímá data z `rmqmessage`	*RMQGetMsgData - Získat datovou část zprávy RMQ na str 559*
Přikázat a zapnout přerušení pro určený datový typ	*IRMQMessage - Přikazuje přerušení RMQ pro datový typ na str 288*
`RMQ Slot`	*rmqslot - Číslo identity klienta RMQ na str 1570*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.143 RobName - Získat jméno TCP robotu

Použití

`RobName` (*Robot Name*) se používá k získání jména TCP robotu v některé programové úloze. Jestliže úloha nekontroluje žádný TCP robot, tato funkce vrátí prázdný řetězec.

Základní příklady

Následující příklad názorně ukazuje funkci`RobName`.

Viz také *Další příklady na str 1304*.

Příklad 1

```
VAR string my_robot;
...
my_robot := RobName();
IF my_robot="" THEN
  TPWrite "This task does not control any TCP robot";
ELSE
  TPWrite "This task controls TCP robot with name "+ my_robot;
ENDIF
```

Zapsat na FlexPendant jméno TCP robotu, který je řízen z této programové úlohy. Jestliže není řízen žádný TCP robot, zapište, že tato úloha neřídí žádný robot.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Jméno mechanické jednotky pro TCP robot, který je kontrolován z této programové úlohy. Vrátit prázdný řetězec, jestliže není kontrolován žádný TCP robot.

Další příklady

Více příkladů jak používat instrukci `RobName` je názorně uvedeno dole.

Příklad 1

```
VAR string my_robot;
...
IF TaskRunRob() THEN
    my_robot := RobName();
    TPWrite "This task controls robot with name "+ my_robot;
ENDIF
```

Jestliže tato programová úloha kontroluje jakýkoliv TCP robot, zapsat jméno tohoto TCP robotu do FlexPendantu.

Syntaxe

```
    RobName '(' ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Zkontrolovat, jestli úloha provozuje nějaký TCP robot	*TaskRunRob - Zkontrolujte, jestli úloha řídí nějaký robot na str 1358*
Zkontrolovat, jestli úloha provozuje nějakou mechanickou jednotku	*TaskRunMec - Zkontrolujte, jestli úloha řídí nějakou mechanickou jednotku na str 1357*
Získat jména mechanických jednotek v systému	*GetNextMechUnit - Získat jméno a data pro mechanické jednotky na str 1172*
Funkce s řetězci	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.144 RobOS - Zkontrolovat, jestli vykonávání je na RC nebo VC

Použití

RobOS (*Robot Operating System*) může být použit pro kontrolu, jestli vykonávání je prováděno na řadiči robotu RC nebo na virtuálním řadiči VC.

Základní příklady

Následující příklad názorně ukazuje funkci RobOS.

Příklad 1

```
IF RobOS() THEN
  ! Execution statements in RC
ELSE
  ! Execution statements in VC
ENDIF
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: bool

TRUE, jestliže vykonávání běží na řadiči robotu RC, jinak FALSE.

Syntaxe

```
RobOS '(' ')'
```

Funkce s vrácenou hodnotou datového typu bool.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.145 Round - Zaokrouhlit numerickou hodnotu

Použití

Round se používá pro zaokrouhlení numerické hodnoty na určený počet desetinných míst nebo na hodnotu celého čísla.

Základní příklady

Následující příklady názorně ukazují funkci Round.

Příklad 1

```
VAR num val;
val := Round(0.3852138\Dec:=3);
```

Proměnné val je dána hodnota 0.385.

Příklad 2

```
val := Round(0.3852138\Dec:=1);
```

Proměnné val je dána hodnota 0.4.

Příklad 3

```
val := Round(0.3852138);
```

Proměnné val je dána hodnota 0.

Příklad 4

```
val := Round(0.3852138\Dec:=6);
```

Proměnné val je dána hodnota 0.385214.

Vratná hodnota (Vrátit hodnotu)

Datový typ: num

Numerická hodnota zaokrouhlená na určený počet desetinných míst.

Argumenty

```
Round ( Val [\Dec])
```

Val

Value

Datový typ: num

Numerická hodnota, která bude zaokrouhlena.

[\Dec]

Decimals

Datový typ: num

Počet desetinných míst.

Jestliže určený počet desetinných míst je 0 nebo když je argument vypuštěn, hodnota bude zaokrouhlena na celé číslo.

Počet desetinných míst nesmí být záporný nebo větší než dostupná přesnost pro numerické hodnoty.

Max počet desetinných míst, která mohou být použita, je 6.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.145 Round - Zaokrouhlit numerickou hodnotu

RobotWare - OS

Pokračování

Syntaxe

```
Round '('
  [ Val ':=' ] <expression (IN) of num>
  [ \Dec ':=' <expression (IN) of num> ] ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Zaokrouhlení hodnoty	*Trunc - Zaokrouhluje numerickou hodnotu na str 1376*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.146 RoundDnum - Zaokrouhlit numerickou hodnotu

Použití

`RoundDnum` se používá pro zaokrouhlení numerické hodnoty na určený počet desetinných míst nebo na hodnotu celého čísla.

Základní příklady

Následující příklady názorně ukazují funkci`RoundDnum`.

Příklad 1

```
VAR dnum val;
val := RoundDnum(0.3852138754655357\Dec:=3);
```

Proměnné `val` je dána hodnota `0.385`.

Příklad 2

```
val := RoundDnum(0.3852138754655357\Dec:=1);
```

Proměnné `val` je dána hodnota `0.4`.

Příklad 3

```
val := RoundDnum(0.3852138754655357);
```

Proměnné `val` je dána hodnota `0`.

Příklad 4

```
val := RoundDnum(0.3852138754655357\Dec:=15);
```

Proměnné `val` je dána hodnota `0.385213875465536`.

Příklad 5

```
val := RoundDnum(1000.3852138754655357\Dec:=15);
```

Proměnné `val` je dána hodnota `1000.38521387547`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Numerická hodnota zaokrouhlená na určený počet desetinných míst.

Argumenty

```
RoundDnum ( Val [\Dec])
```

`Val`

Value

Datový typ: `dnum`

Numerická hodnota, která bude zaokrouhlena.

`[\Dec]`

Decimals

Datový typ: `num`

Počet desetinných míst.

Jestliže určený počet desetinných míst je 0 nebo když je argument vypuštěn, hodnota bude zaokrouhlena na celé číslo.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.146 RoundDnum - Zaokrouhlit numerickou hodnotu

RobotWare - OS

Pokračování

Počet desetinných míst nesmí být záporný nebo větší než dostupná přesnost pro numerické hodnoty.

Max počet desetinných míst, která mohou být použita, je 15.

Syntaxe

```
RoundDnum '('
  [ Val ':=' ] <expression (IN) of dnum>
  [ \Dec ':=' <expression (IN) of num> ] ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Zaokrouhlení hodnoty	*Round - Zaokrouhlit numerickou hodnotu na str 1307*
Zaokrouhlení hodnoty	*Trunc - Zaokrouhluje numerickou hodnotu na str 1376*
Zaokrouhlení hodnoty	*TruncDnum - Zaokrouhluje numerickou hodnotu na str 1378*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.147 RunMode - Přečíst provozní režim

Použití

RunMode*Running Mode* se používá pro čtení aktuálního provozního režimu programové úlohy.

Základní příklady

Následující příklad názorně ukazuje funkciRunMode.

Příklad 1

```
IF RunMode() = RUN_CONT_CYCLE THEN
...
ENDIF
```

Programová sekce je vykonávána pouze pro trvalý nebo cyklický běh.

Vratná hodnota (Vrátit hodnotu)

Datový typ: symnum

Aktuální provozní režim je definován tak, jak je popsáno v tabulce dole.

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
0	RUN_UNDEF	Nedefinovaný provozní režim
1	RUN_CONT_CYCLE	Trvalý nebo cyklický provozní režim
2	RUN_INSTR_FWD	Instrukce pro režim běhu dopředu
3	RUN_INSTR_BWD	Instrukce pro režim běhu dozadu
4	RUN_SIM	Simulovaný provozní režim. Nebyl dosud uvolněn.
5	RUN_STEP_MOVE	Pohybové instrukce v režimu běhu dopředu a logické instrukce v trvalém režimu běhu

Argumenty

```
RunMode ( [ \Main] )
```

[\Main]

Datový typ: switch

Vrátit aktuální režim pro úlohu, jestliže se jedná o pohybovou úlohu. Jestliže byl použit v nepohybové úloze, vrátí aktuální režim pohybové úlohy, ke které je připojena nepohybová úloha.

Jestliže tento argument je vynechán, vrácená hodnota vždy zrcadlí aktuální běžící režim pro programovou úlohu, která vykonává funkci RunMode.

Syntaxe

```
RunMode '('
  ['\' Main] ')'
```

Funkce s vrácenou hodnotou datového typu symnum.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.147 RunMode - Přečíst provozní režim

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Čtení provozního režimu	*OpMode - Přečíst provozní režim na str 1241*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.148 SafetyControllerGetChecksum - Získat kontrolní součet pro uživatelsky konfigurovaný soubor

Použití

`SafetyControllerGetChecksum` se používá k získání kontrolního součtu bezpečnostního řadiče pro uživatelsky konfigurovaný soubor.

Základní příklady

Následující příklad názorně ukazuje funkci`SafetyControllerGetChecksum`.

Příklad 1

```
VAR string mystring;
...
mystring:=SafetyControllerGetChecksum();
```

Získat kontrolní součet pro uživatelsky konfigurovaný soubor a uložit ho do proměnné `mystring`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Kontrolní součet pro uživatelskou konfiguraci.

Syntaxe

```
SafetyControllerGetChecksum '(' ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
`SafetyControllerGetUserChecksum`	*SafetyControllerGetUserChecksum - Získat kontrolní součet pro chráněné parametry na str 1315*
`SafetyControllerGetSWVersion`	*SafetyControllerGetSWVersion - Získat verzi firmwaru bezpečnostního řadiče na str 1314*
`SafetyControllerSyncRequest`	*SafetyControllerSyncRequest - Iniciace hardwarové synchronizační procedury na str 577*
Bezpečnostní konfiguraceSafeMove	*Application manual - Functional safety and SafeMove*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.149 SafetyControllerGetSWVersion - Získat verzi firmwaru bezpečnostního řadiče

Použití

`SafetyControllerGetSWVersion` - se používá k získání verze firmwaru bezpečnostního řadiče.

Základní příklady

Následující příklad názorně ukazuje funkci`SafetyControllerGetSWVersion`.

Příklad 1

```
VAR string mystring;
...
mystring:=SafetyControllerGetSWVersion();
```

Získat verzi firmwaru bezpečnostního řadiče a uložit ji do proměnné `mystring`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Verze firmwaru bezpečnostního řadiče. Řetězec s „VC“ je vrácen, jestliže tato funkce je použita na virtuálním řadiči.

Syntaxe

```
SafetyControllerGetSWVersion '(' ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
`SafetyControllerGetUserChecksum`	*SafetyControllerGetUserChecksum - Získat kontrolní součet pro chráněné parametry na str 1315*
`SafetyControllerGetSWVersion`	*SafetyControllerGetSWVersion - Získat verzi firmwaru bezpečnostního řadiče na str 1314*
`SafetyControllerSyncRequest`	*SafetyControllerSyncRequest - Iniciace hardwarové synchronizační procedury na str 577*
Bezpečnostní konfiguraceSafeMove	*Application manual - Functional safety and SafeMove*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.150 SafetyControllerGetUserChecksum - Získat kontrolní součet pro chráněné parametry

Použití

`SafetyControllerGetUserChecksum` se používá k získání kontrolního součtu bezpečnostního řadiče pro oblast s chráněnými parametry v souboru uživatelské konfigurace.

Základní příklady

Následující příklad názorně ukazuje funkci`SafetyControllerGetUserChecksum`.

Příklad 1

```
VAR string mystring;
...
mystring:=SafetyControllerGetUserChecksum();
```

Získat kontrolní součet pro oblast s chráněnými parametry v souboru uživatelské konfigurace a uložit ho do proměnné `mystring`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Kontrolní součet pro oblast s chráněnými parametry.

Syntaxe

```
SafetyControllerGetUserChecksum '(' ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
`SafetyControllerGetChecksum`	*SafetyControllerGetChecksum - Získat kontrolní součet pro uživatelsky konfigurovaný soubor na str 1313*
`SafetyControllerGetSWVersion`	*SafetyControllerGetSWVersion - Získat verzi firmwaru bezpečnostního řadiče na str 1314*
`SafetyControllerSyncRequest`	*SafetyControllerSyncRequest - Iniciace hardwarové synchronizační procedury na str 577*
Bezpečnostní konfiguraceSafeMove	*Application manual - Functional safety and SafeMove*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.151 Sin - Vypočítává hodnotu sinu

Použití

`Sin`(*Sine*)se používá k výpočtu hodnoty sinu z hodnoty úhlu.

Základní příklady

Následující příklad názorně ukazuje funkci`Sin`.

Příklad 1

```
VAR num angle;
VAR num value;
...
...
value := Sin(angle);
```

`value` získá hodnotu sinu `angle`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Hodnota sinu, rozsah [-1, 1] .

Argumenty

```
Sin (Angle)
```

`Angle`

Datový typ: `num`

Hodnota úhlu vyjádřená ve stupních.

Syntaxe

```
Sin '('
  [Angle ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.152 SinDnum - Vypočítává hodnotu sinu

Použití

`SinDnum` (*Sine dnum*) se používá k výpočtu hodnoty sinu z hodnoty úhlu na datových typech `dnum`.

Základní příklady

Následující příklad názorně ukazuje funkci`SinDnum`.

Příklad 1

```
VAR dnum angle;
VAR dnum value;
...
...
value := SinDnum(angle);
```

`value` získá hodnotu sinu `angle`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Hodnota sinu, rozsah [-1, 1] .

Argumenty

```
SinDnum (Angle)
```

`Angle`

Datový typ: `dnum`

Hodnota úhlu vyjádřená ve stupních.

Syntaxe

```
SinDnum '('
  [Angle ':='] <expression (IN) of dnum> ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.153 SocketGetStatus - Získat aktuální stav socketu

Použití

`SocketGetStatus` vrací aktuální stav socketu.

Základní příklady

Následující příklad názorně ukazuje funkci`SocketGetStatus`.

Viz také *Další příklady na str 1318*.

Příklad 1

```
VAR socketdev socket1;
VAR socketstatus state;
...
SocketCreate socket1;
state := SocketGetStatus( socket1 );
```

Status socketu `SOCKET_CREATED` bude uložen do proměnné `state`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `socketstatus`

Aktuální stav socketu.

Pouze předdefinované symbolické konstanty typu `socketstatus` se mohou používat pro kontrolu stavu.

Argumenty

```
SocketGetStatus( Socket )
```

`Socket`

Datový typ: `socketdev`

Proměnná socketu, jehož stav nás zajímá.

Vykonávání programu

Funkce vrací jeden z následujících předdefinovaných stavů socketstatus:

`SOCKET_CREATED`, `SOCKET_CONNECTED`, `SOCKET_BOUND`, `SOCKET_LISTENING` nebo `SOCKET_CLOSED`.

Další příklady

Následující příklad názorně ukazuje funkci`SocketGetStatus`.

Příklad 1

```
VAR socketstatus status;
VAR socketdev my_socket;
...
SocketCreate my_socket;
SocketConnect my_socket, "192.168.0.1", 1025;
! A lot of RAPID code
status := SocketGetStatus( my_socket );
!Check which instruction that was executed last, not the state of
!the socket
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
IF status = SOCKET_CREATED THEN
  TPWrite "Instruction SocketCreate has been executed";
ELSEIF status = SOCKET_CLOSED THEN
  TPWrite "Instruction SocketClose has been executed";
ELSEIF status = SOCKET_BOUND THEN
  TPWrite "Instruction SocketBind has been executed";
ELSEIF status = SOCKET_LISTENING THEN
  TPWrite "Instruction SocketListen or SocketAccept has been
       executed";
ELSEIF status = SOCKET_CONNECTED THEN
  TPWrite "Instruction SocketConnect, SocketReceive or SocketSend
       has been executed";
ELSE
  TPWrite "Unknown socket status";
ENDIF
```

Socket klienta je vytvořen a připojen ke vzdálenému počítači. Předtím, než je socket použit v instrukci `SocketSend`, stav socketu je zkontrolován, jestli je stále připojen.

Omezení

Stav socketu může být změněn pouze vykonáním instrukce socketu RAPID. Například, jestliže socket je připojen a později je spojení přerušeno, nebude to hlášeno funkcí `SocketGetStatus`. Namísto toho bude vrácena chyba, když je socket použit v instrukci `SocketSend` or `SocketReceive`.

Syntaxe

```
SocketGetStatus '('
  [ Socket ':=' ] < variable (VAR) of socketdev > ')'
```

Funkce s vrácenou hodnotou datového typu `socketstatus`.

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Připojte se ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*
Odeslat data ke vzdálenému počítači	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Přijmout data od vzdáleného počítače	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Zavřít zásuvku	*SocketClose - Zavřít socket na str 659*
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.154 SocketPeek - Test přítomnosti dat na socketu

Použití

SocketPeek se používá k testování přítomnosti dat na socketu. Vrací počet bajtů, který může být přijat na určeném socketu.

Základní příklady

Následující příklad názorně ukazuje funkciSocketPeek.

Příklad 1

```
VAR socketdev socket1;
VAR socketdev client_socket;
VAR num peek_value;
...
SocketCreate socket1;
SocketBind socket1, "192.168.0.1", 1025;
SocketListen socket1;
SocketAccept socket1, client_socket;
..
peek_value := SocketPeek( client_socket );
IF peek_value >= 64 THEN
  SocketReceive client_socket \Str := str_data \ReadNoOfBytes:=64;
  ..
ELSE
  ! Not enough data to receive. Do something else.
ENDIF
```

Nejprve je vytvořen serverový socket a navázán na port 1015 na síťové adrese řadiče 192.168.0.1. Potom je použit SocketPeek pro kontrolu, jestli je k dispozici 64 bajtů pro příjem na socketu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: num

Počet bajtů dostupných na určeném socketu.

Argumenty

```
SocketPeek( Socket )
```

Socket

Datový typ: socketdev

Proměnná socketu, který by měl být zobrazen.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná ERRNO bude nastavena na:

ERR_SOCK_CLOSED	Socket je zavřen. Přerušené spojení.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Všechny sockety jsou zavřeny po restartu po výpadku napájení. Tento problém může být vyřešen obnovou po chybě.

Max velikost dat, která mohou být přijata v jednom volání, je omezena na 1024 bajtů. Proto max hodnota, která může být vrácena od `SocketPeek`, je 1024.

Syntaxe

```
SocketPeek '('
  [ Socket ':=' ] < variable (VAR) of socketdev > ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Připojte se ke vzdálenému počítači (pouze klient)	*SocketConnect - Připojit ke vzdálenému počítači na str 661*
Odeslat data ke vzdálenému počítači	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Odeslat data ke vzdálenému počítači	*SocketSendTo - Odeslat data ke vzdálenému počítači na str 681*
Zavřít zásuvku	*SocketClose - Zavřít socket na str 659*
Navázat socket (pouze server)	*SocketBind - Navázat socket k mé IP adrese a portu na str 657*
Poslechová spojení (pouze server)	*SocketListen - Poslouchat příchozí spojení na str 666*
Přijměte spojení (pouze server)	*SocketAccept - Přijmout příchozí spojení na str 654*
Získat aktuální stav zásuvky	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Příklad klientské socketové aplikace	*SocketSend - Odeslat data ke vzdálenému počítači na str 677*
Přijmout data od vzdáleného počítače	*SocketReceive - Přijmout data od vzdáleného počítače na str 668*
Přijmout data od vzdáleného počítače	*SocketReceiveFrom - Přijmout data od vzdáleného počítače na str 673*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.155 Sqrt - Vypočítává hodnotu druhé odmocniny

Použití

`Sqrt` (*Square root*) se používá pro výpočet hodnoty druhé odmocniny.

Základní příklady

Následující příklad názorně ukazuje funkci`Sqrt`.

Příklad 1

```
VAR num x_value;
VAR num y_value;
...
...
y_value := Sqrt( x_value);
```

`y-value` získá hodnotu druhé odmocniny `x_value`, to znamená √(`x_value`).

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Hodnota druhé odmocniny (√).

Argumenty

```
Sqrt (Value)
```

`Value`

Datový typ: `num`

Hodnota argumentu pro druhou odmocninu, tj. √`value`.

`Value` musí být ≥ 0.

Omezení

Vykonání funkce `Sqrt(x)` vykáže chybu, jestliže `x` < 0.

Syntaxe

```
Sqrt '('
  [Value ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Vypočítat hodnotu druhé odmocniny z numerické hodnoty dnum	*SqrtDnum - Vypočítává hodnotu druhé odmocniny na str 1323*
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.156 SqrtDnum - Vypočítává hodnotu druhé odmocniny

Použití

`SqrtDmum` (*Square root dnum)* se používá pro výpočet hodnoty druhé odmocniny.

Základní příklady

Následující příklad názorně ukazuje funkci`SqrtDnum`.

Příklad 1

```
VAR dnum x_value;
VAR dnum y_value;
...
...
y_value := SqrtDnum(x_value);
```

`y_value` získá hodnotu druhé odmocniny `x_value`, to znamená √(`x_value`).

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Hodnota druhé odmocniny (√).

Argumenty

```
SqrtDnum (Value)
```

`Value`

Datový typ: `dnum`

Hodnota argumentu pro druhou odmocninu, tj. √`value`.

`Value` musí být ≥ 0.

Omezení

Vykonání funkce `Sqrt(x)` vykáže chybu, jestliže `x` < 0.

Syntaxe

```
SqrtDnum '('
  [ Value ':=' ] < expression (IN) of dnum > ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Vypočítat hodnotu druhé odmocniny z numerické hodnoty num	*Sqrt - Vypočítává hodnotu druhé odmocniny na str 1322*
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.157 STCalcForce - Vypočítává sílu hrotu pro servo nástroj

Použití

`STCalcForce` se používá k výpočtu síly hrotu pro servo nástroj. Tato funkce se používá, například, ke zjištění max přípustné síly hrotu pro servo nástroj.

Základní příklady

Následující příklad názorně ukazuje funkci`STCalcForce`.

Příklad 1

```
VAR num tip_force;
tip_force := STCalcForce(gun1, 7);
```

Vypočítat sílu hrotu, když požadovaný točivý moment motoru je `7` Nm.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Vypočítaná síla hrotu [N].

Argumenty

```
STCalcForce ToolName MotorTorque
```

`ToolName`

Datový typ: `string`

Jméno mechanické jednotky.

`MotorTorque`

Datový typ: `num`

Požadovaný točivý moment motoru [Nm].

Řešení chyb

Jestliže jméno určeného servo nástroje není konfigurovaný servo nástroj, potom je systémová proměnná `ERRNO` nastavena na `ERR_NO_SGUN`.

Chyba může být zpracována v handleru chyb RAPID.

Syntaxe

```
STCalcForce '('
  [ ToolName ':=' ] < expression (IN) of string > ','
  [ MotorTorque ':=' ] < expression (IN) of num > ';'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Otevřít servo nástroj	*STOpen - Otevřít servo nástroj na str 734*
Zavřít servo nástroj	*STClose - Zavřít servo nástroj na str 717*
Vypočítat točivý moment motoru	*STCalcTorque - Vypočítat točivý moment motoru pro servo nástroj na str 1325*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.158 STCalcTorque - Vypočítat točivý moment motoru pro servo nástroj

Použití

`STCalcTorque` se používá k výpočtu točivého momentu motoru pro servo nástroj. Tato funkce se používá, například, když se provádí kalibrace síly.

Základní příklady

Následující příklad názorně ukazuje funkci`STCalcTorque`.

Příklad 1

```
VAR num curr_motortorque;
curr_motortorque := STCalcTorque( gun1, 1000);
```

Vypočítat točivý moment motoru, když požadovaná síly hrotu je `1000` N.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Vypočítaný točivý moment motoru [Nm].

Argumenty

```
STCalcTorque ToolName TipForce
```

`ToolName`

Datový typ: `string`

Jméno mechanické jednotky.

`TipForce`

Datový typ: `num`

Požadovaná síla hrotu [N].

Řešení chyb

Jestliže jméno určeného servo nástroje není konfigurovaný servo nástroj, potom je systémová proměnná `ERRNO` nastavena na `ERR_NO_SGUN`.

Chyba může být zpracována v handleru chyb RAPID.

Syntaxe

```
STCalcTorque '('
  [ ToolName ':=' ] < expression (IN) of string > ','
  [ TipForce ':=' ] < expression (IN) of num > ';'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Otevřít servo nástroj	*STOpen - Otevřít servo nástroj na str 734*
Zavřít servo nástroj	*STClose - Zavřít servo nástroj na str 717*
Vypočítat sílu hrotu	*STCalcForce - Vypočítává sílu hrotu pro servo nástroj na str 1324*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.159 STIsCalib - Testuje, jestli servo nástroj je kalibrován

Použití

`STIsCalib` se používá k testování, jestli servo nástroj je kalibrován - tj. kontroluje, jestli hroty pistole jsou kalibrovány nebo synchronizovány.

Základní příklady

Následující příklady názorně ukazují funkci`STIsCalib`.

Příklad 1

```
IF STIsCalib(gun1\sguninit) THEN
  ...
ELSE
  !Start the gun calibration
  STCalib gun1\TipChg;
ENDIF
```

Příklad 2

```
IF STIsCalib(gun1\sgunsynch) THEN
  ...
ELSE
  !Start the gun calibration to synchronize the gun position with
       the revolution counter
  STCalib gun1\ToolChg;
ENDIF
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže testovaný nástroj je kalibrován - tj. vzdálenost mezi hroty nástroje je kalibrována, nebo jestli je testovaný nástroj synchronizován, tj. pozice hrotů nástroje je synchronizována s počítadlem otáček nástroje.

`FALSE`, jestliže testovaný nástroj není kalibrován nebo synchronizován.

Argumenty

```
STIsCalib ToolName [\sguninit] | [\sgunsynch]
```

`ToolName`

Datový typ: `string`

Jméno mechanické jednotky.

`[ \sguninit ]`

Datový typ: `switch`

Tento argument se používá pro kontrolu, jestli pozice pistole je iniciována a kalibrována.

`[ \sgunsynch ]`

Datový typ: `switch`

Tento argument se používá pro kontrolu, jestli pozice pistole je synchronizována v počítadlem otáček.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
STIsCalib '('
  [ ToolName ':=' ] < expression (IN) of string >
  [ '\' sguninit ] | [ '\'sgunsynch ] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Kalibrování servo nástroje	*STCalib - Kalibrovat servo nástroj na str 713*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.160 STIsClosed - Testuje, jestli servo nástroj je zavřen

Použití

`STIsClosed` se používá k testování, jestli servo nástroj je zavřen.

Základní příklady

Následující příklady názorně ukazují funkci`STIsClosed`.

Příklad 1

```
IF STIsClosed(gun1) THEN
  !Start the weld process
  Set weld_start;
ELSE
  ...
ENDIF
```

Zkontrolovat, jestli pistole je zavřena nebo nikoliv.

Příklad 2

```
STClose "sgun", 1000, 3 \Conc;
WHILE NOT(STIsClosed("sgun"\RetThickness:=thickness)) DO
  WaitTime 0.1;

ENDWHILE
IF thickness > max_thickness THEN...
```

Spusťte zavírání pistole pojmenované `sgun`. Pokračujte okamžitě s další instrukcí, ve které program čeká na zavření pistole. Přečtěte dosaženou hodnotu tloušťky, když instrukce `STIsClosed` vrátila `TRUE`.

Příklad 3

Příklady neplatných kombinací:

```
STClose "sgun", 1000, 3 \RetThickness:=thickness \Conc;
WHILE NOT(STIsClosed("sgun"\RetThickness:=thickness_2)) DO;
...
```

Zavřete pistoli. Parametrová tloušťka **nebude** držet žádnou platnou hodnotu, jelikož je použit přepínač `\Conc`. Čekejte, až se pistole zavře. Když je pistole zavřena a `STIsClosed` vrací `TRUE`, parametrová tloušťka 2 bude držet platnou hodnotu, jelikož přepínač `\Conc` byl použit pro `STClose`.

```
STClose "sgun", 1000, 3 \RetThickness:=thickness;
WHILE NOT(STIsClosed("sgun"\RetThickness:=thickness_2)) DO;
...
```

Zavřete pistoli. Parametrová tloušťka bude držet platnou hodnotu, když pistole byla zavřena, jelikož přepínač `\Conc` není použit. Parametrová tloušťka 2 nebude držet žádnou platnou hodnotu, jelikož přepínač `\Conc` nebyl použit v instrukci `STClose`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

TRUE, jestliže testovaný nástroje je zavřen, tj. bylo dosaženo požadované síly hrotu.

FALSE, jestliže testovaný nástroj není zavřen.

Argumenty

```
STIsClosed ToolName [\RetThickness]
```

`ToolName`

Datový typ: `string`

Jméno mechanické jednotky.

`[\RetThickness]`

Datový typ: `num`

Dosažená tloušťka [mm].

POZOR! Platné pouze při použití `\Conc` v předchozí instrukci `STClose`.

Syntaxe

```
STIsClosed '('
  [ ToolName ':=' ] < expression (IN) of string > ')'
  ['\' RetThickness ':=' < variable or persistent (INOUT) of num
       > ] ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Otevřít servo nástroj	*STOpen - Otevřít servo nástroj na str 734*
Zavřít servo nástroj	*STClose - Zavřít servo nástroj na str 717*
Otestovat, jestli servo nástroj je otevřen	*STIsOpen - Testuje, jestli servo nástroj je otevřen na str 1331*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.161 STIsIndGun - Testuje, jestli servo nástroj je v nezávislém režimu

Použití

`STIsIndGun` se používá k testování, jestli servo nástroj je v nezávislém režimu.

Základní příklady

Následující příklad názorně ukazuje funkci`STIsIndGun`.

Příklad 1

```
IF STIsIndGun(gun1) THEN
  ! Start the gun calibration
  STCalib gun1\TipChg;
ELSE
  ...
ENDIF
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže testovaný nástroj je v nezávislém režimu - tj. pistole může být posunuta nezávisle na pohybech robotu.

`FALSE`, jestliže testovaný nástroj *not* v nezávislém režimu.

Argumenty

```
STIsIndGun ToolName
```

`ToolName`

Datový typ: `string`

Jméno mechanické jednotky.

Syntaxe

```
STIsIndGun '('
  [ ToolName ':=' ] < expression (IN) of string > ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Kalibrování servo nástroje	*STCalib - Kalibrovat servo nástroj na str 713*
Nastavování pistole do nezávislého režimu	*STIndGun - Nastavuje pistoli do nezávislého režimu na str 722*
Resetování pistole z nezávislého režimu	*STIndGunReset - Resetuje pistoli z nezávislého režimu na str 724*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.162 STIsOpen - Testuje, jestli servo nástroj je otevřen

Použití

`STIsOpen` se používá k testování, jestli servo nástroj je otevřen.

Základní příklady

Následující příklady názorně ukazují funkci`STIsOpen`.

Příklad 1

```
IF STIsOpen(gun1) THEN
  !Start the motion
  MoveL ...
ELSE
  ...
ENDIF
```

Zkontrolovat, jestli pistole je otevřena nebo nikoliv.

Příklad 2

```
STCalib "sgun" \TipWear \Conc;
WHILE NOT(STIsOpen("sgun") \RetTipWear:=tipwear \RetPosAdj:=posadj)
     DO;
WaitTime 0.1;

ENDWHILE
```

IF tipwear > 20...

IF posadj > 25...

Proveďte kalibraci opotřebení hrotu. Čekejte, až bude pistole `sgun` otevřena. Přečtěte hodnoty opotřebení hrotu a pozičního nastavení.

Příklad 3

Příklady neplatných kombinací:

```
STCalib "sgun" \TipWear \RetTipWear:=tipwear_1 \Conc;
WHILE NOT(STIsOpen("sgun") \RetTipWear:=tipwear_2) DO;
  WaitTime 0.1;

ENDWHILE
```

Spusťte kalibraci opotřebení hrotu. Parametr `tipwear_1` **nebude** držet žádnou platnou hodnotu, jelikož je použit přepínač `\Conc`. Když je kalibrace hotová a `STIsOpen` vrací `TRUE`, parametr `tipwear_2` bude držet platnou hodnotu.

```
STCalib "sgun" \TipWear \RetTipWear:=tipwear_1;

WHILE NOT(STIsOpen("sgun") \RetTipWear:=tipwear_2) DO;
  WaitTime 0.1;

ENDWHILE
```

Proveďte kalibraci opotřebení hrotu. Parametr `tipwear_1` bude držet platnou hodnotu, jelikož není použit přepínač `\Conc`. Když `STIsOpen` vrací `TRUE`, parametr

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.162 STIsOpen - Testuje, jestli servo nástroj je otevřen

Servo Tool Control

Pokračování

`tipwear_2` nebude držet žádnou platnou hodnotu, jelikož přepínač `\Conc` nebyl použit v `STCalib`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže testovaný nástroj je otevřen, tj. rameno nástroje je v naprogramované otevřené poloze.

`FALSE`, jestliže testovaný nástroj není otevřen.

Argumenty

```
STIsOpen ToolName [\RetTipWear] [\RetPosAdj]
```

`ToolName`

Datový typ: `string`

Jméno mechanické jednotky.

`[\RetTipWear]`

Datový typ: `num`

Dosažené opotřebení hrotu [mm].

POZOR! Platné pouze při použití `\Conc` v předchozí instrukci `STCalib` a jestliže `STIsOpen` vrací `TRUE`.

`[\RetPosAdj]`

Datový typ: `num`

Nastavení pozice od poslední kalibrace [mm].

POZOR! Platné pouze při použití `\Conc` v předchozí instrukci `STCalib` a jestliže `STIsOpen` vrací `TRUE`.

Syntaxe

```
STIsOpen '('
  [ ToolName ':=' ] < expression (IN) of string > ')'
  [ '\' RetTipWear ':=' < variable or persistent(INOUT) of num >
       ] ';'
  [ '\' RetPosAdj ':=' < variable or persistent(INOUT) of num > ]
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Otevřít servo nástroj	*STOpen - Otevřít servo nástroj na str 734*
Zavřít servo nástroj	*STClose - Zavřít servo nástroj na str 717*
Otestovat, jestli servo nástroj je zavřen	*STIsClosed - Testuje, jestli servo nástroj je zavřen na str 1328*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.163 StrDigCalc - Aritmetické operace s datovým typem stringdig

Použití

`StrDigCalc` se používá k provádění aritmetických operací (+, -, *, /, %) na dvou kladných číselných řetězcích stejným způsobem jako numerické aritmetické operace na kladných hodnotách celého čísla.

Tato funkce může zpracovat kladná celá čísla nad 8 388 608 s přesnou reprezentací.

Základní příklady

Následující příklad názorně ukazuje funkci`StrDigCalc`.

Viz také *Další příklady na str 1334*.

Příklad 1

```
res := StrDigCalc(str1, OpAdd, str2);
```

Pro `res` je přidělen výsledek přičítací operace na hodnotách reprezentovaných digitálními řetězci `str1` a `str2`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `stringdig`

`stringdig` se používá k reprezentaci velkých kladných celých čísel v řetězci pouze s číslicemi.

Tento datový typ je zaveden, protože datový typ `num` nemůže zpracovat kladná celá čísla nad 8 388 608 s přesnou reprezentací.

Argumenty

```
StrDigCalc (StrDig1 Operation StrDig2)
```

`StrDig1`

String Digit 1

Datový typ: `stringdig`

Řetězec reprezentující hodnotu kladného celého čísla.

`Operation`

Arithmetic operator

Datový typ: `opcalc`

Definuje aritmetickou operaci, která bude provedena na dvou číselných řetězcích. Následující aritmetické operace datového typu `opcalc` mohou být použity; `OpAdd`, `OpSub`, `OpMult`, `OpDiv` a `OpMod`.

`StrDig2`

String Digit 2

Datový typ: `stringdig`

Řetězec reprezentující hodnotu kladného celého čísla.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.163 StrDigCalc - Aritmetické operace s datovým typem stringdig
RobotWare - OS
Pokračování

Vykonávání programu

Tato funkce bude:

- Kontrolovat pouze čísla 0...9 v `StrDig1` a `StrDig2`
- Převádět dva digitální řetězce na `long integers`
- Provádět aritmetickou operaci na dvou `long integers`
- Převádět výsledek z `long integer` na `stringdig`

Další příklady

Následující příklady názorně ukazují funkci`StrDigCalc`.

Příklad 1

```
res := StrDigCalc(str1, OpSub, str2);
```

Pro `res` je přidělen výsledek odečítací operace na hodnotách reprezentovaných digitálními řetězci `str1` a `str2`.

Příklad 2

```
res := StrDigCalc(str1, OpMult, str2);
```

Pro `res` je přidělen výsledek násobicí operace na hodnotách reprezentovaných digitálními řetězci `str1` a `str2`.

Příklad 3

```
res := StrDigCalc(str1, OpDiv, str2);
```

Pro `res` je přidělen výsledek dělicí operace na hodnotách reprezentovaných digitálními řetězci `str1` a `str2`.

Příklad 4

```
res := StrDigCalc(str1, OpMod, str2);
```

Pro `res` je přidělen výsledek modulus operace na hodnotách reprezentovaných digitálními řetězci `str1` a `str2`.

Řešení chyb

Následující chyby mohou být zpracovány v chybovém handleru RAPID.

Kód chyby	Popis
ERR_INT_NOTVAL	Vložit hodnoty nikoliv jen číslice nebo modulus od nuly
ERR_INT_MAXVAL	Vložit hodnoty nad 4294967295
ERR_CALC_OVERFLOW	Výsledek mimo rozsah 0...4294967295
ERR_CALC_NEG	Záporný odečet, tj. StrDig2 > StrDig1
ERR_CALC_DIVZERO	Dělení nulou

Omezení

`StrDigCalc` akceptuje pouze řetězce, které obsahují číslice (znaky 0...9). Všechny ostatní znaky v `stringdig` budou mít výsledek chybu.

Tato funkce může zpracovat pouze kladná celá čísla do 4 294 967 295.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
StrDigCalc '('
  [ StrDig1 ':=' ] < expression (IN) of stringdig > ','
  [ Operation ':=' ] < expression (IN) of opcalc > ','
  [ StrDig2 ':=' ] < expression (IN) of stringdig > ')'‘
```

Funkce s vrácenou hodnotou datového typu `stringdig`.

Související informace

Pro informace o	Viz
Řetězce pouze s číslicemi.	*stringdig - Řetězec pouze s číslicemi na str 1598*
Aritmetické operátory.	*opcalc - Arithmetic Operator na str 1540*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.164 StrDigCmp - Porovnat dva řetězce pouze s číslicemi

Použití

`StrDigCmp` se používá k porovnání dvou kladných číselných řetězců stejným způsobem, jako numerické srovnání kladných celých čísel.

Tato funkce může zpracovat kladná celá čísla nad 8 388 608 s přesnou reprezentací.

Základní příklady

Následující příklady názorně ukazují funkci`StrDigCmp`.

Příklad 1

```
VAR stringdig digits1 := "1234";
VAR stringdig digits2 := "1256";
VAR bool is_equal;
is_equal := StrDigCmp(digits1, EQ, digits2);
```

Proměnná `is_equal` bude nastavena na `FALSE`, protože numerická hodnota `1234` není rovna `1256`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

TRUE, jestliže daná podmínka je splněna, FALSE, jestliže nikoliv.

Argumenty

```
StrDigCmp (StrDig1 Relation StrDig2)
```

`StrDig1`

String Digit 1

Datový typ: `stringdig`

První řetězec pouze s číslicemi, který bude numericky porovnán.

`Relation`

Datový typ: `opnum`

Definuje, jak porovnat dva číselné řetězce. Následující předdefinované konstanty datového typu `opnum` mohou být použity: `LT, LTEQ, EQ, NOTEQ, GTEQ` nebo `GT`.

`StrDig2`

String Digit 2

Datový typ: `stringdig`

Druhý řetězec pouze s číslicemi, který bude numericky porovnán.

Vykonávání programu

Tato funkce bude:

- Zkontrolovat, že jsou použita pouze čísla 0...9 v `StrDig1` a `StrDig2`
- Převádět dva digitální řetězce na `long integers`
- Numericky porovnat dva `long integers`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Řešení chyb

Následující chyby mohou být zpracovány v chybovém handleru RAPID.

Kód chyby	Popis
ERR_INT_NOTVAL	Vložit hodnoty, nejen číslice
ERR_INT_MAXVAL	Hodnota nad 4294967295

Omezení

`StrDigCmp` akceptuje pouze řetězce, které obsahují číslice (znaky 0...9). Všechny ostatní znaky v `stringdig` budou mít výsledek chybu.

Tato funkce může zpracovat pouze kladná celá čísla do 4 294 967 295.

Syntaxe

```
StrDigCmp '('
  [ StrDig1 ':=' ] < expression (IN) of stringdig > ','
  [ Relation ':=' ] < expression (IN) of opnum > ','
  [ StrDig2 ':=' ] < expression (IN) of stringdig > ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Řetězec pouze s číslicemi	*stringdig - Řetězec pouze s číslicemi na str 1598*
Srovnávací operátory	*opnum - Srovnávací operátor na str 1541*
Časová informace souboru	*FileTimeDnum - Získat časovou informaci o souboru na str 1158*
Čas poslední modifikace načteného modulu	*ModTimeDnum - Získat čas modifikace souboru pro načtený modul na str 1227*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.165 StrFind - Hledá znak v řetězci

Použití

`StrFind` (*String Find*) se používá pro vyhledání v řetězci, se začátkem na určené pozici, znaku, který přísluší k určené sadě znaků.

Základní příklady

Následující příklad názorně ukazuje funkci`StrFind`.

Příklad 1

```
VAR num found;
found := StrFind("Robotics",1,"aeiou");
```

Proměnné `found` je dána hodnota 2.

```
found := StrFind("Robotics",1,"aeiou"\NotInSet);
```

Proměnné `found` je dána hodnota 1.

```
found := StrFind("IRB 6400",1,STR_DIGIT);
```

Proměnné `found` je dána hodnota 5.

```
found := StrFind("IRB 6400",1,STR_WHITE);
```

Proměnné `found` je dána hodnota `4`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Pozice prvního znaku na nebo za určenou pozicí, který přísluší k určené sadě. Jestliže takový znak není nalezen, délka řetězce +1 je vrácena.

Argumenty

```
StrFind (Str ChPos Set [\NotInSet])
```

`Str`

String

Datový typ: `string`

Řetězec, ve kterém se bude hledat.

`ChPos`

Character Position

Datový typ: `num`

Pozice počátečního znaku. Jestliže pozice je mimo řetězec, bude vyvolána chyba za běhu.

`Set`

Datový typ: `string`

Sada znaků, proti kterým se bude testovat. Viz také *Předdefinovaná data na str 1339*.

`[\NotInSet]`

Datový typ: `switch`

Hledat znaky, které nejsou v sadě znaků uvedených v `Set`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
StrFind '('
  [ Str ':=' ] <expression (IN) of string> ','
  [ ChPos ':=' ] <expression (IN) of num> ','
  [ Set ':=' ] <expression (IN) of string>
  ['\' NotInSet ] ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Předdefinovaná data

Řada předdefinovaných řetězcových konstant je dostupná v systému a může se používat společně s funkcemi řetězce.

Název	Znaková sada
STR_DIGIT	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
STR_UPPER	<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï | 1)| Ñ | Ò | Ó | Ô | Õ | Ö | Ø | Ù | Ú | Û | Ü | 2) | 3)
STR_LOWER	<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø | ù | ú | û | ü | 2) | 3) | ß | ÿ-
STR_WHITE	<blank character> ::=

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.166 StrLen - Získá délku řetězce

Použití

`StrLen`*String Length* se používá pro zjištění aktuální délky řetězce.

Základní příklady

Následující příklad názorně ukazuje funkci`StrLen`.

Příklad 1

```
VAR num len;
len := StrLen("Robotics");
```

Proměnné `len` je dána hodnota `8`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Počet znaků v řetězci (>=0).

Argumenty

```
StrLen (Str)
```

`Str`

String

Datový typ: `string`

Řetězec, do kterého se započítá počet znaků.

Syntaxe

```
StrLen '('
  [ Str ':=' ] <expression (IN) of string> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - RAPID - Instrukce, funkce a datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.167 StrMap - Mapuje řetězec

Použití

`StrMap` (*String Mapping*) se používá k vytvoření kopie řetězce, do kterého jsou všechny znaky přeloženy podle určeného mapování.

Základní příklady

Následující příklady názorně ukazují funkci`StrMap`.

Příklad 1

```
VAR string str;
str := StrMap("Robotics","aeiou","AEIOU");
```

Proměnné `str` je dána hodnota `RObOtIcs`.

Příklad 2

```
str := StrMap("Robotics",STR_LOWER, STR_UPPER);
```

Proměnné `str` je dána hodnota `ROBOTICS`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Řetězec vytvořený přeložením znaků do určeného řetězce, jak je určeno řetězci „od“ a „do“. Každý znak z určeného řetězce, který je nalezen v řetězci „od“, je nahrazen znakem na odpovídající pozici v řetězci „do“. Znaky, pro které není definováno žádné mapování, jsou zkopírovány beze změny do výsledného řetězce.

Argumenty

```
StrMap ( Str FromMap ToMap)
```

`Str`

String

Datový typ: `string`

Řetězec k přeložení.

`FromMap`

Datový typ: `string`

Indexová část mapování. Viz také *Předdefinovaná data na str 1342*.

`ToMap`

Datový typ: `string`

Hodnotová část mapování. Viz také *Předdefinovaná data na str 1342*.

Syntaxe

```
StrMap '('
  [ Str ':=' ] <expression (IN) of string> ','
  [ FromMap ':=' ] <expression (IN) of string> ','
  [ ToMap ':=' ] <expression (IN) of string> ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.167 StrMap - Mapuje řetězec

RobotWare - OS

Pokračování

Předdefinovaná data

Řada předdefinovaných řetězcových konstant je dostupná v systému a může se používat společně s funkcemi řetězce.

Název	Znaková sada
STR_DIGIT	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
STR_UPPER	<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï | 1)| Ñ | Ò | Ó | Ô | Õ | Ö | Ø | Ù | Ú | Û | Ü | 2) | 3)
STR_LOWER	<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø | ù | ú | û | ü | 2) | 3) | ß | ÿ-
STR_WHITE	<blank character> ::=

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.168 StrMatch - Hledat vzor v řetězci

Použití

`StrMatch` (*String Match*) se používá pro vyhledání určeného vzoru v řetězci, se začátkem na určené pozici.

Základní příklady

Následující příklad názorně ukazuje funkci`StrMatch`.

Příklad 1

```
VAR num found;

found := StrMatch("Robotics",1,"bo");
```

Proměnné `found` je dána hodnota `3`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Pozice znaku prvního podřetězce na nebo za určenou pozicí, která je totožná s řetězcem určeného vzoru. Jestliže takový podřetězec není nalezen, délka řetězce +1 je vrácena.

Argumenty

```
StrMatch (Str ChPos Pattern)
```

`Str`

String

Datový typ: `string`

Řetězec, ve kterém se bude hledat.

`ChPos`

Character Position

Datový typ: `num`

Pozice počátečního znaku. Jestliže pozice je mimo řetězec, bude vyvolána chyba za běhu.

`Pattern`

Datový typ: `string`

Řetězec vzoru, který se bude hledat.

Syntaxe

```
StrMatch '('
  [ Str ':=' ] <expression (IN) of string> ','
  [ ChPos ':=' ] <expression (IN) of num> ','
  [ Pattern ':=' ] <expression (IN) of string> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.168 StrMatch - Hledat vzor v řetězci

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.169 StrMemb - Kontroluje, jestli znak patří do sady

Použití

`StrMemb` (*String Member*) se používá ke kontrole, jestli určený znak v řetězci přísluší k určené sadě znaků.

Základní příklady

Následující příklad názorně ukazuje funkci`StrMemb`.

Příklad 1

```
VAR bool memb;
memb := StrMemb("Robotics",2,"aeiou");
```

Proměnné `memb` je dána hodnota `TRUE`, jelikož `o` je členem sady `"aeiou"`.

```
memb := StrMemb("Robotics",3,"aeiou");
```

Proměnné `memb` je dána hodnota `FALSE`, jelikož `b` není členem sady `"aeiou"`.

```
memb := StrMemb("S-721 68 VÄSTERÅS",3,STR_DIGIT);
```

Proměnné `memb` je dána hodnota `TRUE`, jelikož `7` je členem sady `STR_DIGIT`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže znak na určené pozici v určeném řetězci přísluší do určené sady znaků.

Argumenty

```
StrMemb (Str ChPos Set)
```

`Str`

String

Datový typ: `string`

Řetězec, ve kterém se bude provádět kontrola.

`ChPos`

Character Position

Datový typ: `num`

Pozice znaku ke kontrole. Jestliže pozice je mimo řetězec, bude vyvolána chyba za běhu.

`Set`

Datový typ: `string`

Sada znaků, proti které se bude testovat.

Syntaxe

```
StrMemb '('
  [ Str ':=' ] <expression (IN) of string> ','
  [ ChPos ':=' ] <expression (IN) of num> ','
  [ Set ':=' ] <expression (IN) of string> ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.169 StrMemb - Kontroluje, jestli znak patří do sady

RobotWare - OS

Pokračování

Předdefinovaná data

Řada předdefinovaných řetězcových konstant je dostupná v systému a může se používat společně s funkcemi řetězce.

Název	Znaková sada
STR_DIGIT	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
STR_UPPER	<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | À | Á | Â | Ã | Î | Ï | 1)| Ñ | Ò | Ó | Ô | Õ | Ö | Ø | Ù | Ú | Û | Ü | 2) | 3)
STR_LOWER	<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø | ù | ú | û | ü | 2) | 3) | ß | ÿ-
STR_WHITE	<blank character> ::=

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.170 StrOrder - Kontroluje, jestli řetězce jsou seřazeny

Použití

`StrOrder` (*String Order*) porovnává dva řetězce (znak po znaku) a vrací boolean označující, jestli jsou dva řetězce v pořadí podle určené sekvence pořadí znaků.

Základní příklady

Následující příklady názorně ukazují funkci`StrOrder`.

Příklad 1

```
VAR bool le;

le := StrOrder("FIRST","SECOND",STR_UPPER);
```

Proměnné `le` je dána hodnota `TRUE`, protože "F" přichází před "S“ v sekvenci řazení znaků `STR_UPPER`.

Příklad 2

```
VAR bool le;

le := StrOrder("FIRST","FIRSTB",STR_UPPER);
```

Proměnné `le` je dána hodnota `TRUE`, protože ""`FIRSTB`"" má další znak v sekvenci řazení znaků (žádný znak v porovnání s `"B"`).

Příklad 3

```
VAR bool le;

le := StrOrder("FIRSTB","FIRST",STR_UPPER);
```

Proměnné `le` je dána hodnota `FALSE`, protože ""`FIRSTB`"" má další znak v sekvenci řazení znaků (`"B"` nebyl porovnáván s žádným znakem).

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže první řetězec přichází před druhým řetězcem (`Str1` <= `Str2`), když jsou znaky řazeny podle určení.

Všechny znaky, které nejsou zahrnuty v definovaném pořadí, jsou považovány za následující existujících znaků.

Argumenty

```
StrOrder ( Str1 Str2 Order)
```

`Str1`

String 1

Datový typ: `string`

Hodnota prvního řetězce.

`Str2`

String 2

Datový typ: `string`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.170 StrOrder - Kontroluje, jestli řetězce jsou seřazeny

RobotWare - OS

Pokračování

Hodnota druhého řetězce.

`Order`

Datový typ: `string`

Sekvence znaků, která definuje pořadí. Viz také *Předdefinovaná data na str 1348*.

Syntaxe

```
StrOrder '('
  [ Str1 ':=' ] <expression (IN) of string> ','
  [ Str2 ':=' ] <expression (IN) of string> ','
  [ Order ':=' ] <expression (IN) of string> ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Předdefinovaná data

Řada předdefinovaných řetězcových konstant je dostupná v systému a může se používat společně s funkcemi řetězce.

Název	Znaková sada
STR_DIGIT	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
STR_UPPER	<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï | 1)| Ñ | Ò | Ó | Ô | Õ | Ö | Ø | Ù | Ú | Û | Ü | 2) | 3)
STR_LOWER	<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø | ù | ú | û | ü | 2) | 3) | ß | ÿ-
STR_WHITE	<blank character> ::=

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.171 StrPart - Hledá část řetězce

Použití

`StrPart` (*String Part)* se používá k hledání části řetězce jako nového řetězce.

Základní příklady

Následující příklad názorně ukazuje funkci`StrPart`.

Příklad 1

```
VAR string part;
part := StrPart("Robotics",1,5);
```

Proměnné `part` je dána hodnota `"Robot"`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Podřetězec určeného řetězce, který má určenou délku a začíná na určené znakové pozici.

Argumenty

```
StrPart (Str ChPos Len)
```

`Str`

String

Datový typ: `string`

Řetězec, ve kterém má být nalezena část.

`ChPos`

Character Position

Pozice počátečního znaku. Jestliže pozice je mimo řetězec, bude vyvolána chyba za běhu.

`Len`

Délka

Datový typ: `num`

Délka části řetězce. Chyba za běhu bude vyvolána, jestliže délka je záporná nebo větší než délka řetězce, nebo když podřetězec je (částečně) mimo řetězec.

Syntaxe

```
StrPart '('
[ Str ':=' ] <expression (IN) of string> ','
[ ChPos ':=' ] <expression (IN) of num> ','
[ Len ':=' ] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.171 StrPart - Hledá část řetězce

RobotWare - OS

Pokračování

Pro informace o	Viz
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.172 StrToByte - Převádí řetězec na bajtová data

Použití

`StrToByte` (*String To Byte*) se používá pro převod řetězce s definovaným formátem bajtových dat na bajtová data.

Základní příklady

Následující příklad názorně ukazuje funkci`StrToByte`.

Příklad 1

```
VAR string con_data_buffer{5} := ["10", "AE", "176", "00001010",
     "A"];
VAR byte data_buffer{5};
data_buffer{1} := StrToByte(con_data_buffer{1});
```

Obsah komponentu pole `data_buffer{1}` bude 10 desítkových po funkci `StrToByte ....`

```
data_buffer{2} := StrToByte(con_data_buffer{2}\Hex);
```

Obsah komponentu pole `data_buffer{2}` bude 174 desítkových po funkci `StrToByte ....`

```
data_buffer{3} := StrToByte(con_data_buffer{3}\Okt);
```

Obsah komponentu pole `data_buffer{3}` bude 126 desítkových po funkci `StrToByte ... .`

```
data_buffer{4} := StrToByte(con_data_buffer{4}\Bin);
```

Obsah komponentu pole `data_buffer{4}` bude 10 desítkových po funkci `StrToByte ... .`

```
data_buffer{5} := StrToByte(con_data_buffer{5}\Char);
```

Obsah komponentu pole `data_buffer{5}` bude 65 desítkových po funkci `StrToByte ....`

Vratná hodnota (Vrátit hodnotu)

Datový typ: `byte`

Výsledek převodní operace v desítkové reprezentaci.

Argumenty

```
StrToByte (ConStr [\Hex] | [\Okt] | [\Bin] | [\Char])
```

`ConStr`

Convert String

Datový typ: `string`

Řetězcová data, která budou převedena.

Jestliže je vypuštěn volitelný argument přepínače, řetězec, který bude převeden, má formát `decimal` (**Dec**).

`[\Hex]`

Hexadecimal

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.172 StrToByte - Převádí řetězec na bajtová data

RobotWare - OS

Pokračování

Řetězec, který bude převeden, má formát `hexadecimal`.

`[\Okt]`

Octal

Datový typ: `switch`

Řetězec, který bude převeden, má formát `octal`.

`[\Bin]`

Binary

Datový typ: `switch`

Řetězec, který bude převeden, má formát `binary`.

`[\Char]`

Character

Datový typ: `switch`

Řetězec, který bude převeden, má znakový formát `ASCII`.

Omezení

Podle formátu řetězce k převedení jsou platná následující řetězcová data:

Formát	Délka řetězce	Rozsah
Dec .....: ’0’ - ’9’	3	"0" - "255"
Hex .....: ’0’ - ’9’, ’a’ -’f’, ’A’ - ’F’	2	"0" - "FF"
Okt .....: ’0’ - ’7’	3	"0" - "377"
Bin .....: ’0’ - ’1’	8	"0" - "11111111"
Char ....: Jakýkoliv znak ASCII	1	Jeden znak ASCII

Znakové kódy RAPID (například, “\07” pro kontrolní znak BEL) se mohou používat jako argumenty v `ConStr`.

Syntaxe

```
StrToByte '('
  [ConStr ':=' ] <expression (IN) of string>
  ['\' Hex ] | ['\' Okt] | ['\' Bin] | ['\' Char]
  ')'
```

Funkce s vrácenou hodnotou datového typu `byte`.

Související informace

Pro informace o	Viz
Konvertovat bajt na řetězcová data	*ByteToStr - Převádí byte na řetězcová data na str 1077*
Ostatní bitové (bajtové) funkce	*Technická referenční příručka - Přehled RAPID*
Ostatní řetězcové funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.173 StrToVal - Převádí řetězec na hodnotu

Použití

`StrToVal` (*String To Value*) se používá k převodu řetězce na hodnotu jakéhokoliv datového typu.

Základní příklady

Následující příklad názorně ukazuje funkci`StrToVal`.

Viz také *Další příklady na str 1353*.

Příklad 1

```
VAR bool ok;
VAR num nval;
ok := StrToVal("3.85",nval);
```

Proměnné `ok` je dána hodnota `TRUE`, a `nval` je dána hodnota `3.85`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže požadovaný převod byl úspěšný, jinak `FALSE`.

Argumenty

```
StrToVal ( Str Val )
```

`Str`

String

Datový typ: `string`

Řetězcová hodnota obsahující literální data s formátem odpovídajícím datovému typu použitému v argumentu `Val`. Platný formát jako pro literální skupiny RAPID.

`Val`

Value

Datový typ: `ANYTYPE`

Jméno proměnné nebo perzistentu jakéhokoliv datového typu k uložení výsledku převodu.

Mohou se používat všechny typy hodnotových dat se strukturou atomický, záznam, komponent záznamu, pole nebo element pole. Data jsou nezměněna, jestliže požadovaný převod selhal, protože formát neodpovídal datům použitým v argumentu `Str`.

Další příklady

Více příkladů funkce `StrToVal` je názorně uvedeno dole.

Příklad 1

```
VAR string str15 := "[600, 500, 225.3]";
VAR bool ok;
VAR pos pos15;

ok := StrToVal(str15,pos15);
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.173 StrToVal - Převádí řetězec na hodnotu

RobotWare - OS

Pokračování

Proměnné `ok` je dána hodnota `TRUE` a proměnné `pos15` je dána hodnota, která je určena v řetězci `str15`.

Syntaxe

```
StrToVal '('
  [ Str ':=' ] <expression (IN) of string> ','
  [ Val ':=' ] <var or pers (INOUT) of ANYTYPE>
  ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.174 Tan - Vypočítává hodnotu tangenty

Použití

`Tan` (*Tangent*) se používá k výpočtu hodnoty tangenty z hodnoty úhlu.

Základní příklady

Následující příklad názorně ukazuje funkci`Tan`.

Příklad 1

```
VAR num angle;
VAR num value;
...
...
value := Tan(angle);
```

`value` získá hodnotu tangenty z `angle`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Hodnota tangenty.

Argumenty

```
Tan (Angle)
```

`Angle`

Datový typ: `num`

Hodnota úhlu vyjádřená ve stupních.

Syntaxe

```
Tan '('
  [Angle ':='] <expression (IN) of num>
  ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Arkustangens vrátí hodnotu v rozsahu [-180,180]	*ATan2 - Vypočítá hodnotu arkustangens2 na str 1046*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.175 TanDnum - Vypočítává hodnotu tangenty

Použití

`TanDnum` (*Tangent*) se používá k výpočtu hodnoty tangenty z hodnoty úhlu na datových typech `dnum`.

Základní příklady

Následující příklad názorně ukazuje funkci`TanDnum`.

Příklad 1

```
VAR dnum angle;
VAR dnum value;
...
...
value := TanDnum(angle);
```

`value` získá hodnotu tangenty z `angle`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Hodnota tangenty.

Argumenty

```
TanDnum (Angle)
```

`Angle`

Datový typ: `dnum`

Hodnota úhlu vyjádřená ve stupních.

Syntaxe

```
TanDnum '('
  [Angle ':='] <expression (IN) of dnum>
  ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Arkustangens vrátí hodnotu v rozsahu [-180,180]	*ATan2Dnum - Vypočítá hodnotu arkustangens2 na str 1047*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.176 TaskRunMec - Zkontrolujte, jestli úloha řídí nějakou mechanickou jednotku

Použití

`TaskRunMec` se používá ke kontrole, jestli programová úloha řídí nějakou mechanickou jednotku (robot s TCP nebo manipulátor bez TCP).

Základní příklady

Následující příklad názorně ukazuje funkci`TaskRunMec`.

Příklad 1

```
VAR bool flag;
...
flag := TaskRunMec( );
```

Jestliže aktuální úloha řídí některou mechanickou jednotku, příznak `flag` bude `TRUE`, jinak `FALSE`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Jestliže aktuální úloha řídí některou mechanickou jednotku, vrácená hodnota bude `TRUE`, jinak `FALSE`.

Vykonávání programu

Zkontrolujte, jestli aktuální programová úloha řídí některou mechanickou jednotku.

Syntaxe

```
TaskRunMec '(' ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Zkontrolujte, jestli úloha řídí nějaký robot	*TaskRunRob - Zkontrolujte, jestli úloha řídí nějaký robot na str 1358*
Aktivace/deaktivace mechanických jednotek	*ActUnit - Aktivuje mechanickou jednotku na str 24* *DeactUnit - Deaktivuje mechanickou jednotku na str 150*
Konfigurace mechanických jednotek	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.177 TaskRunRob - Zkontrolujte, jestli úloha řídí nějaký robot

Použití

`TaskRunRob` se používá ke kontrole, jestli programová jednotka řídí nějaký robot (mechanickou jednotku s TCP).

Základní příklady

Následující příklad názorně ukazuje funkci`TaskRunRob`.

Příklad 1

```
VAR bool flag;
...
flag := TaskRunRob( );
```

Jestliže aktuální úloha řídí nějaký robot, `flag` bude nastaven na TRUE, jinak FALSE.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Jestliže aktuální úloha řídí nějaký robot, vrácená hodnota bude TRUE, jinak FALSE.

Vykonávání programu

Zkontrolujte, jestli aktuální programová úloha řídí nějaký robot.

Syntaxe

```
TaskRunRob '(' ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Zkontrolujte, jestli úloha řídí nějakou mechanickou jednotku	*TaskRunMec - Zkontrolujte, jestli úloha řídí nějakou mechanickou jednotku na str 1357*
Aktivace/deaktivace mechanických jednotek	*ActUnit - Aktivuje mechanickou jednotku na str 24* *DeactUnit - Deaktivuje mechanickou jednotku na str 150*
Konfigurace mechanických jednotek	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.178 TasksInSync - Vrací počet synchronizovaných úloh

Použití

`TasksInSync` se používá k vyhledání počtu synchronizovaných úloh.

Základní příklady

Následující příklad názorně ukazuje funkci`TaskInSync`.

Příklad 1

```
VAR tasks tasksInSyncList{6};
...

PROC main ()
  VAR num noOfSynchTasks;
  ...
  noOfSynchTasks:= TasksInSync (tasksInSyncList);
  TPWrite "No of synchronized tasks = "\Num:=noOfSynchTasks;
ENDPROC
```

Proměnné `noOfSynchTasks` je přidělen počet synchronizovaných úloh a `tasksInSyncList` bude obsahovat jména synchronizovaných úloh. V tomto příkladu je seznam úloh proměnnou, ale může to být také perzistent.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Počet synchronizovaných úloh.

Argumenty

```
TaskInSync (TaskList)
```

`TaskList`

Datový typ: `tasks`

Inout argument, který bude v seznamu úloh (pole) představovat jméno (`string`) programových úloh, které jsou synchronizovány. Seznam úloh může být buď typu `VAR` nebo `PERS`.

Vykonávání programu

Funkce vrací počet synchronizovaných úloh v systému. Jména synchronizovaných úloh jsou uvedena v inout argumentu TaskList. V případech, kde nejsou žádné synchronizované úlohy, bude seznam obsahovat pouze prázdné řetězce.

Omezení

Aktuálně je podporována pouze jedna synch skupina, takže `TasksInSync` vrací počet úloh, které jsou synchronizovány v této skupině.

Syntaxe

```
TasksInSync
  [ TaskList ':=' ] < var or pers array {*} (INOUT) of tasks> ','
```

Funkce s vrácenou hodnotou datového typu `num`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.178 TasksInSync - Vrací počet synchronizovaných úloh

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Určit spolupracující programové úlohy	*tasks - Programové úlohy RAPID na str 1607*
Spustit koordinované synchronizované pohyby	*SyncMoveOn - Spustit koordinované synchronizované pohyby na str 758*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.179 TestAndSet - Otestovat proměnnou a nastavit, pokud není nastavena

Použití

`TestAndSet` se může používat společně s normálním datovým objektem typu `bool` jako binární semafor k vyhledání exkluzivního práva ke konkrétním oblastem kódů RAPID nebo systémovým zdrojům. Funkci je možné používat mezi různými programovými úlohami a různými úrovněmi vykonávání (TRAP nebo událostní rutiny) v rámci stejné programové úlohy.

Příklad zdrojů, které mohou potřebovat ochranu před přístupem ve stejnou dobu:

- Použití některých rutin RAPID s funkčními problémy při vykonávání souběžně.
- Použití FlexPendantu - Protokol operátora

Základní příklady

Následující příklad názorně ukazuje funkci`TestAndSet`.

Viz také *Další příklady na str 1362*.

Příklad 1

Programová úloha `MAIN`:

```
PERS bool tproutine_inuse := FALSE;
...
WaitUntil TestAndSet(tproutine_inuse);
TPWrite "First line from MAIN";
TPWrite "Second line from MAIN";
TPWrite "Third line from MAIN";
tproutine_inuse := FALSE;
```

Programová úloha `BACK1`:

```
PERS bool tproutine_inuse := FALSE;
...
WaitUntil TestAndSet(tproutine_inuse);
TPWrite "First line from BACK1";
TPWrite" Second line from BACK1";
TPWrite "Third line from BACK1";
tproutine_inuse := FALSE;
```

Kvůli zabránění pomíchání řádek v protokolu operátora, jedna z `MAIN` a jedna z `BACK1`), zaručuje použití funkce `TestAndSet`, že všechny tři řádky z každé úlohy nejsou odděleny.

Jestliže programová úloha `MAIN` převezme semafor `TestAndSet(tproutine_inuse)` jako první, potom programová úloha `BACK1` musí čekat, až programová úloha `MAIN` semafor opustí.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE`, jestliže semafor byl převzat mnou (vykonavatel funkce `TestAndSet`), jinak `FALSE`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.179 TestAndSet - Otestovat proměnnou a nastavit, pokud není nastavena

RobotWare - OS

Pokračování

Argumenty

```
TestAndSet Object
```

`Object`

Datový typ: `bool`

Uživatelsky definovaný datový objekt, který bude použit jako semafor. Datovým objektem může být proměnná VAR nebo perzistentní proměnná PERS. Jestliže jsou použity `TestAndSet` mezi odlišnými programovými úlohami, objektem musí být perzistentní proměnná PERS nebo instalovaná proměnná VAR (meziúlohové objekty).

Vykonávání programu

Tato funkce bude v jednou nedělitelném kroku kontrolovat uživatelsky definovanou proměnnou, a jestliže není nastavena, nastaví ji a vrátí TRUE, jinak vrátí FALSE.

```
IF Object = FALSE THEN
  Object := TRUE;
  RETURN TRUE;
ELSE
  RETURN FALSE;
ENDIF
```

Další příklady

Následující příklad názorně ukazuje funkci`TestAndSet`.

Příklad 1

```
LOCAL VAR bool doit_inuse := FALSE;
...
PROC doit(...)
  WaitUntil TestAndSet (doit_inuse);
  ...
  doit_inuse := FALSE;
ENDPROC
```

Jestliže modul je instalovaný vestavěný a sdílený, je možné používat lokální proměnnou modulu pro ochranu přístupu z různých programových úloh ve stejnou dobu.

> **POZNÁMKA**
>
> V tomto případě s instalovaným vestavěnými moduly a při používání perzistentní proměnné jako semaforového objektu: Jestliže vykonávání programu je zastaveno v rutině `doit` a ukazatel programu je posunut na `main`, proměnná `doit_inuse` nebude resetována. Abyste se tomu vyhnuli, resetujte proměnnou `doit_inuse` na FALSE v událostní rutině START.

Syntaxe

```
TestAndSet '('
  [ Object ':=' ] < variable or persistent (INOUT) of bool> ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Čekejte na zrušení nastavení proměnné - potom nastavte (zapsat čekat s kontrolou přerušení)	*WaitTestAndSet - Čekat, až se proměnná změní na FALSE, potom nastavit na str 960*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.180 TestDI - Otestuje nastavení digitálního vstupu

Použití

`TestDI` se používá k otestování nastavení digitálního vstupu.

Základní příklady

Následující příklad názorně ukazuje funkci`TestDI`.

Příklad 1

```
IF TestDI (di2) THEN . . .
```

Jestliže aktuální hodnota signálu `di2` je rovna `1`, potom . . .

```
IF NOT TestDI (di2) THEN . . .
```

Jestliže aktuální hodnota signálu `di2` je rovna `0`, potom . . .

```
WaitUntil TestDI(di1) AND TestDI(di2);
```

Vykonávání programu pokračuje pouze po nastavení vstupu `di1` a vstupu `di2`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

`TRUE` = Aktuální hodnota signálu je rovna 1.

`FALSE` = Aktuální hodnota signálu je rovna 0.

Argumenty

```
TestDI (Signal)
```

`Signal`

Datový typ: `signaldi`

Jméno signálu, který bude otestován.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT` jestliže není žádný kontakt s jednotkou I/O.

`ERR_SIG_NOT_VALID` jestli není přístup k I/O signálu (platí pouze pro sběrnici ICI).

Syntaxe

```
TestDI '('
  [ Signal ':=' ] < variable (VAR) of signaldi > ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Čtení hodnoty digitálního vstupního signálu	*signalxx - Digitální a analogové signály na str 1582*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Čtení hodnoty digitálního výstupního signálu	*DOutput - Čte hodnotu digitálního výstupního signálu na str 1145*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.181 TestSignRead - Přečíst hodnotu testovacího signálu

Použití

`TestSignRead` se používá k přečtení aktuální hodnoty testovacího signálu.

Tato funkce vrací aktuální hodnotu nebo střední hodnotu posledních vzorků, podle specifikace kanálu v instrukci `TestSignDefine`.

Základní příklady

Následující příklad názorně ukazuje funkci`TestSignRead`.

Viz také *Další příklady na str 1367*.

Příklad 1

```
CONST num speed_channel:=1;
VAR num speed_value;
...
TestSignDefine speed_channel, speed, orbit, 1, 0;
...
! During some movements with orbit’s axis 1
speed_value := TestSignRead(speed_channel);
...
TestSignReset;
```

Pro `speed_value` je přidělena střední hodnota posledních 8 vzorků vytvořených každých 0,5 ms testovacího signálu `speed` na kanálu `speed_channel` definovaném jako kanál `1`. Kanál `speed_channel` měří rychlost `axis 1` na mechanické jednotce `orbit`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Numerická hodnota v SI jednotkách na straně motoru pro určený kanál podle definice v instrukci `TestSignDefine`.

Argumenty

```
TestSignRead (Channel)
```

`Channel`

Datový typ: `num`

Kanál číslo 1-12 pro testovací signál, který bude přečten. Stejné číslo se musí použít v definiční instrukci `TestSignDefine`.

Vykonávání programu

Vrací aktuální hodnotu nebo střední hodnotu posledních vzorků, podle specifikace kanálu v instrukci `TestSignDefine`.

Pro předdefinované testovací signály s platnými SI jednotkami pro osy externího manipulátoru, viz datový typ `testsignal`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Následující příklad názorně ukazuje funkci`TestSignRead`.

Příklad 1

```
CONST num torque_channel:=2;
VAR num torque_value;
VAR intnum timer_int;
CONST jointtarget psync := [...];
...
PROC main()
  CONNECT timer_int WITH TorqueTrap;
  ITimer \Single, 0.05, timer_int;
  TestSignDefine torque_channel, torque_ref, IRBP_K, 2, 0.001;
  ...
  MoveAbsJ psync \NoEOffs, v5, fine, tool0;
  ...
  IDelete timer_int;
  TestSignReset;

TRAP TorqueTrap
  IF (TestSignRead(torque_channel) > 6) THEN
    TPWrite "Torque pos = " + ValToStr(CJointT());
    Stop;
  ELSE
    IDelete timer_int;
    CONNECT timer_int WITH TorqueTrap;
    ITimer \Single, 0.05, timer_int;
  ENDIF
ENDTRAP
```

Jestliže reference momentu pro manipulátor `IRBP_K` osa `2` je poprvé větší než `6` Nm na straně motoru během pomalého pohybu k pozici `psync`, pozice spoje je zobrazena na FlexPendantu.

Syntaxe

```
TestSignRead '('
  [ Channel ':='] <expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou typu `num`.

Související informace

Pro informace o	Viz
Testovací signál	*testsignal - Testovací signál na str 1609*
Definovat testovací signál	*TestSignDefine - Definovat testovací signál na str 776*
Resetovat testovací signály	*TestSignReset - Resetovat všechny definice testovacích signálů na str 778*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.182 TextGet - Vzít text ze systémových textových tabulek

Použití

`TextGet` se používá z získání textového řetězce ze systémových textových tabulek.

Základní příklady

Následující příklad názorně ukazuje funkci`TextGet`.

Příklad 1

```
VAR string text1;
...
text1 := TextGet(14, 5);
```

Proměnné `text1` je přidělen text uložený v textovém zdroji `14` a index `5`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Určený text ze systémových textových tabulek.

Argumenty

```
TextGet ( Table Index )
```

`Table`

Datový typ: `num`

Číslo textové tabulky (kladné celé číslo).

`Index`

Datový typ: `num`

Indexové číslo (kladné celé číslo) v textové tabulce.

Řešení chyb

Jestliže tabulka nebo index nejsou platné a není možné získat žádný textový řetězec ze systémových textových tabulek, systémová proměnná `ERRNO` je nastavena na `ERR_TXTNOEXIST`. Vykonávání pokračuje v chybovém handleru.

Syntaxe

```
TextGet '('
  [ Table ':=' ] < expression (IN) of num > ','
  [ Index ':=' ] < expression (IN) of num> ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Získat číslo textové tabulky	*TextTabGet - Získat číslo textové tabulky na str 1372*
Instalovat textovou tabulku	*TextTabInstall - Instalace textové tabulky na str 779*
Formátovat textové soubory	*Technická referenční příručka - RAPID kernel*
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.183 TextTabFreeToUse - Otestovat, jestli je textová tabulka volná

Použití

`TextTabFreeToUse` by se mělo používat k otestování, jestli jméno textové tabulky (řetězec zdroje textu) je volné k použití (není dosud instalováno v systému), to znamená, jestli je možné instalovat textovou tabulku do systému nebo nikoliv.

Základní příklady

Následující příklad názorně ukazuje funkci`TextTabFreeToUse`.

Příklad 1

```
! System Module with Event Routine to be executed at event
! POWER ON, RESET or START

PROC install_text()
  IF TextTabFreeToUse("text_table_name") THEN
    TextTabInstall "HOME:/text_file.eng";
  ENDIF
ENDPROC
```

Při prvním vykonávání událostní rutiny `install_text` funkce `TextTabFreeToUse` vrací `TRUE` a textový soubor `text_file.eng` je instalován do systému. Potom mohou být instalované textové řetězce získány ze systému do RAPIDu funkcemi `TextTabGet` a `TextGet`.

Příště, když je událostní rutina `install_text` vykonávána, funkce `TextTabFreeToUse` vrací `FALSE` a instalace se **neopakuje**.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Tato funkce vrací:

- `TRUE`, jestliže textová tabulka **není** dosud instalována v systému
- `FALSE`, jestliže textová tabulka je již instalována v systému

Argumenty

```
TextTabFreeToUse ( TableName )
```

`TableName`

Datový typ: `string`

Jméno textové tabulky (řetězec s max. 80 znaky). Podívejte se na `<text_resource>`: v *Referenční příručce RAPID - RAPID Kernel*, sekce *Textové soubory*. Řetězec `text_resource` je jméno textové tabulky.

Omezení

Omezení pro instalaci textových tabulek (testové zdroje) v systému:

- Není možné instalovat stejnou textovou tabulku do systému více než jedenkrát
- Není možné odinstalovat (uvolnit) jednoduchou textovou tabulku ze systému. Jediným způsobem, jak odinstalovat textové tabulky ze systému, je restartovat

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

řadič pomocí režimu restartu **Resetovat systém**. Všechny textové tabulky (definované systémem a uživatelem) budou potom odinstalovány.

Syntaxe

```
TextTabFreeToUse '('
  [ TableName ':=' ] < expression (IN) of string > ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Instalovat textovou tabulku	*TextTabInstall - Instalace textové tabulky na str 779*
Formáty textových souborů	*Technická referenční příručka - RAPID kernel*
Získat číslo textové tabulky	*TextTabGet - Získat číslo textové tabulky na str 1372*
Vzít text ze systémových textových tabulek	*TextGet - Vzít text ze systémových textových tabulek na str 1368*
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.184 TextTabGet - Získat číslo textové tabulky

Použití

`TextTabGet` se používá k získání čísla textové tabulky z uživatelsky definované textové tabulky během provozu.

Základní příklady

Následující příklady názorně ukazují funkci`TextTabGet`.

Oba příklady používají novou textovou tabulku pojmenovanou `deburr_part1` pro uživatelsky definované texty. Nová textová tabulka má jméno souboru `deburr.eng`.

```
# deburr.eng - USERS deburr_part1 english text description file
#
# DESCRIPTION:
# Users text file for RAPID development
#
deburr_part1::
0:
RAPID S4: Users text table deburring part1
1:
Part 1 is not in pos
2:
Identity of worked part: XYZ
3:
Part error in line 1
#
# End of file
```

Příklad 1

```
VAR num text_res_no;
...
text_res_no := TextTabGet("deburr_part1");
```

Proměnné `text_res_no` je přiděleno číslo textové tabulky pro definovanou textovou tabulku `deburr_part1`.

Příklad 2

```
ErrWrite TextGet(text_res_no, 1), TextGet(text_res_no, 2);
```

Zpráva se ukládá do protokolu robotu. Zpráva se také zobrazuje na displeji FlexPendantu. Zprávy budou převzaty z textové tabulky `deburr_part1` :

Part 1 is not in pos

Identity of worked part: XYZ

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Číslo textové tabulky z definované textové tabulky.

Argumenty

```
TextTabGet ( TableName )
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`TableName`

Datový typ: `string`

Jméno textové tabulky.

Syntaxe

```
TextTabGet '('
  [ TableName '=' ] < expression (IN) of string > ';)'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Vzít text ze systémových textových tabulek	*TextGet - Vzít text ze systémových textových tabulek na str 1368*
Instalovat textovou tabulku	*TextTabInstall - Instalace textové tabulky na str 779*
Formátovat textové soubory	*Technická referenční příručka - RAPID kernel*
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.185 TriggDataValid - Zkontrolujte, jestli obsah proměnné triggdata je platný

Použití

Funkce `TriggDataValid` se používá pro kontrolu, jestli proměnná `triggdata` je platná. Platná proměnná `triggdata` je proměnná, která byla dříve použita v programu v jedné z instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO` k určení podmínek a aktivity spouštěče.

Základní příklady

Následující příklad názorně ukazuje funkci`TriggDataValid`.

Příklad 1

```
VAR triggdata trigg_array{25};
...
PROC MyTriggProcL(robtarget myrobt, \VAR triggdata T1 \VAR triggdata
     T2 \VAR triggdata T3)
  VAR num triggcnt:=2;
  ! Reset entire trigg_array array before using it
  FOR i FROM 1 TO 25 DO
    TriggDataReset trigg_array{i};
  ENDFOR
  TriggEquip trigg_array{1}, 10 \Start, 0 \DOp:=do1, SetValue:=1;
  TriggEquip trigg_array{2}, 40 \Start, 0 \DOp:=do2, SetValue:=1;
  ! Check if optional argument is present,
  ! and if any trigger condition has been setup in T1
  IF Present(T1) AND TriggDataValid(T1) THEN
    ! Copy actual trigger condition to trigg_array
    TriggDataCopy trigg_array{triggcnt}, T1;
    Incr triggcnt;
  ENDIF
  IF Present(T2) AND TriggDataValid(T2) THEN
    Incr triggcnt;
    TriggDataCopy trigg_array{triggcnt}, T2;
  ENDIF
  IF Present(T3) AND TriggDataValid(T3) THEN
    Incr triggcnt;
    TriggDataCopy trigg_array{triggcnt}, T3;
  ENDIF
  TriggL p1, v500, trigg_array, z30, tool2;
  ...
```

Procedura `MyTriggProcL` nahoře používá instrukci `TriggDataValid` ke kontrole, jestli jsou nějaká platná data používána ve volitelných argumentech `T1`, `T2` a `T3`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Tato funkce vrací:

- TRUE, jestliže proměnná je platná, tj. jedna z instrukcí `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` nebo `TriggRampAO` byla použita k určení podmínek a aktivity spouštěče.
- FALSE, jestliže proměnná nebyla použita při nastavování jakékoliv podmínky a aktivity spouštěče.

Argumenty

```
TriggDataValid TriggData
```

`TriggData`

Datový typ: `triggdata`

Proměnná `triggdata` pro kontrolu platnosti.

Syntaxe

```
TriggDataValid
  [TriggData ':=' ] < variable (VAR) of triggdata > ';'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Lineární pohyb se spouštěči	*TriggL - Lineární pohyby robotu s událostmi na str 839*
Pohyb spoje se spouštěči	*TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Kruhový pohyb se spouštěči	*TriggC - Kruhový pohyb robotu s událostmi na str 796*
Definice spouštěčů	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814* *TriggInt - Definuje přerušení se vztahem k pozici na str 820* *TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804* *TriggRampAO - Definovat událost rampy AO pevné pozice na dráze na str 861* *TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času na str 868*
Manipulace s `triggdata`	*triggdata - Polohovací události, trigg na str 1618* *TriggDataReset - Resetovat obsah v proměnné trigg-data na str 812* *TriggDataCopy - Kopírovat obsah do proměnné trig-gdata na str 810*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.186 Trunc - Zaokrouhluje numerickou hodnotu

Použití

`Trunc` (*Truncate*) se používá pro zaokrouhlení numerické hodnoty na určený počet desetinných míst nebo na hodnotu celého čísla.

Základní příklady

Následující příklady názorně ukazují funkci`Trunc`.

Příklad 1

```
VAR num val;
val := Trunc(0.3852138\Dec:=3);
```

Proměnné `val` je dána hodnota `0.385`.

Příklad 2

```
reg1 := 0.3852138;
val := Trunc(reg1\Dec:=1);
```

Proměnné `val` je dána hodnota `0.3`.

Příklad 3

```
val := Trunc(0.3852138);
```

Proměnné `val` je dána hodnota `0`.

Příklad 4

```
val := Trunc(0.3852138\Dec:=6);
```

Proměnné `val` je dána hodnota `0.385213`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Numerická hodnota zaokrouhlená na určený počet desetinných míst.

Argumenty

```
Trunc ( Val [\Dec] )
```

`Val`

Value

Datový typ: `num`

Numerická hodnota, která bude zaokrouhlena.

`[\Dec]`

Decimals

Datový typ: `num`

Počet desetinných míst.

Jestliže určený počet desetinných míst je 0 nebo když je argument vypuštěn, hodnota bude zaokrouhlena na celé číslo.

Počet desetinných míst nesmí být záporný nebo větší než dostupná přesnost pro numerické hodnoty.

Max počet desetinných míst, která mohou být použita, je 6.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Syntaxe

```
Trunc '('
  [ Val ':=' ] <expression (IN) of num>
  [ \Dec ':=' <expression (IN) of num> ] ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Zaokrouhlení hodnoty	*Round - Zaokrouhlit numerickou hodnotu na str 1307*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.187 TruncDnum - Zaokrouhluje numerickou hodnotu

Použití

`TruncDnum` (*Truncate dnum*) se používá pro zaokrouhlení numerické hodnoty na určený počet desetinných míst nebo na hodnotu celého čísla.

Základní příklady

Následující příklady názorně ukazují funkci`TruncDnum`.

Příklad 1

```
VAR dnum val;
val := TruncDnum(0.3852138754655357\Dec:=3);
```

Proměnné `val` je dána hodnota `0.385`.

Příklad 2

```
val := TruncDnum(0.3852138754655357\Dec:=1);
```

Proměnné `val` je dána hodnota `0.3`.

Příklad 3

```
val := TruncDnum(0.3852138754655357);
```

Proměnné `val` je dána hodnota `0`.

Příklad 4

```
val := TruncDnum(0.3852138754655357\Dec:=15);
```

Proměnné `val` je dána hodnota `0.385213875465535`.

Příklad 5

```
val := TruncDnum(1000.3852138754655357\Dec:=15);
```

Proměnné `val` je dána hodnota `1000.38521387547`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Numerická hodnota zaokrouhlená na určený počet desetinných míst.

Argumenty

```
TruncDnum ( Val [\Dec] )
```

`Val`

Value

Datový typ: `dnum`

Numerická hodnota, která bude zaokrouhlena.

`[\Dec]`

Decimals

Datový typ: `num`

Počet desetinných míst.

Jestliže určený počet desetinných míst je 0 nebo když je argument vypuštěn, hodnota bude zaokrouhlena na celé číslo.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Počet desetinných míst nesmí být záporný nebo větší než dostupná přesnost pro numerické hodnoty.

Max počet desetinných míst, která mohou být použita, je 15.

Syntaxe

```
TruncDnum '('
  [ Val ':=' ] <expression (IN) of dnum>
  [ \Dec ':=' <expression (IN) of num> ] ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*
Zaokrouhlení hodnoty	*Trunc - Zaokrouhluje numerickou hodnotu na str 1376*
Zaokrouhlení hodnoty	*Round - Zaokrouhlit numerickou hodnotu na str 1307*
Zaokrouhlení hodnoty	*RoundDnum - Zaokrouhlit numerickou hodnotu na str 1309*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.188 Type - Získat jméno datového typu pro proměnnou

Použití

`Type` se používá pro získání jména datového typu pro určenou proměnnou v argumentu `Data`.

Základní příklady

Následující příklady názorně ukazují funkci`Type`.

Příklad 1

```
VAR string rettype;
VAR intnum intnumtype;
...
PROC main()
  rettype := Type(intnumtype);
  TPWrite "Data type name: " + rettype;
```

Vytištěno bude: "Data type name: intnum"

Příklad 2

```
VAR string rettype;
VAR intnum intnumtype;
...
PROC main()
  rettype := Type(intnumtype \BaseName);
  TPWrite "Data type name: " + rettype;
```

Vytištěno bude: "Data type name: num"

Příklad 3

```
VAR string rettype;
VAR num numtype;
...
PROC main()
  rettype := Type(numtype);
  TPWrite "Data type name: " + rettype;
```

Vytištěno bude: "Data type name: num"

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Řetězec se jménem datového typu pro určenou proměnnou v argumentu `Data`.

Argumenty

```
Type (Data [\BaseName])
```

`Data`

Data object name

Datový typ: `anytype`

Jméno proměnné, od které bude získáno jméno datového typu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`[\BaseName]`

Base data type Name

Datový typ: `switch`

Jestliže je použito, potom funkce vrací jméno zásadního datového typu, když určený `Data` je deklarovaného datového typu `ALIAS`.

Syntaxe

```
Type '('
  [ Data ':=' ] < reference (REF) of anytype >
  [ '\' BaseName ] ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Definice typů Alias	*Technická referenční příručka - RAPID kernel* *ALIAS - Přidělování datového typu alias na str 1439*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.189 UIAlphaEntry - Uživatelský vstup

Použití

`UIAlphaEntry` (*User Interaction Alpha Entry*) se používá pro umožnění operátorovi napsat řetězec z dostupného uživatelského zařízení, jako je FlexPendant. Zpráva je napsána pro operátora, který odpoví textovým řetězcem. Řetězec je potom přenesen zpět do programu.

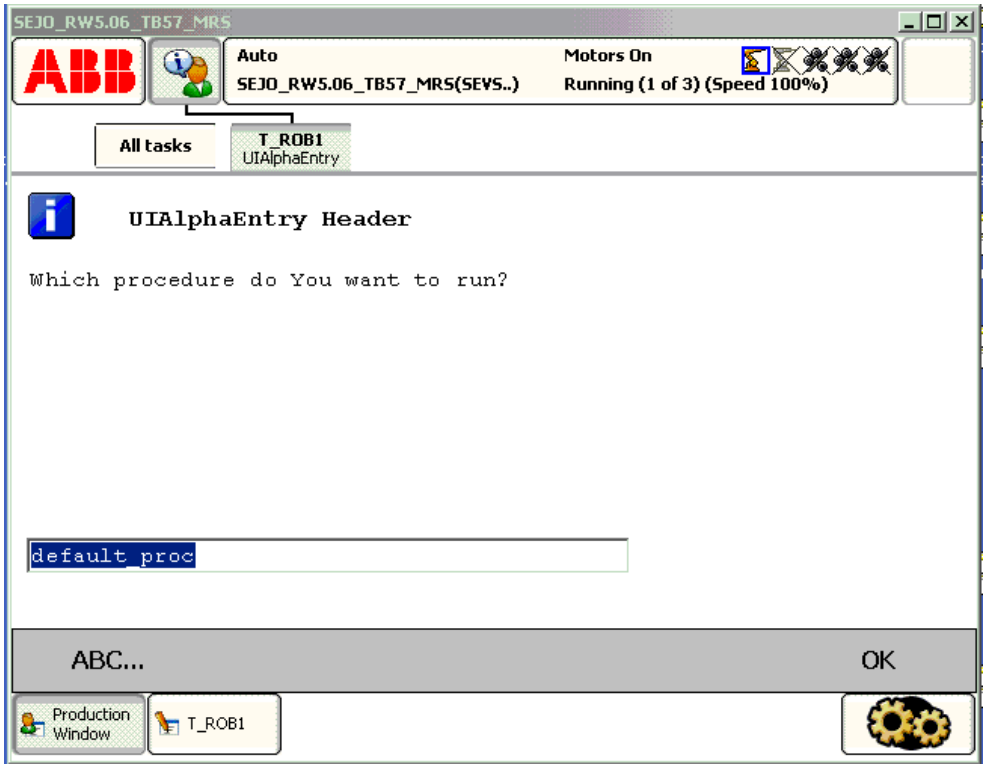
Základní příklady

Následující příklad názorně ukazuje funkci`UIAlphaEntry`.

Viz část *Další příklady na str 1385*.

Příklad 1

```
VAR string answer;
...
answer := UIAlphaEntry(
  \Header:= "UIAlphaEntry Header",
  \Message:= "Which procedure do You want to run?"
  \Icon:=iconInfo
  \InitString:= "default_proc");
%answer%;
```



xx0500002437

Shora uvedené okénko pro textové zprávy s ikonou, hlavičkou, zprávou a init řetězcem jsou napsány na displej FlexPendantu. Uživatel edituje init řetězec nebo napíše nový řetězec s podporovaným Alpha Padem. Vykonávání programu čeká

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

do stisknutí OK a potom je napsaný řetězec vrácen do proměnné `answer`. Program potom volá určenou proceduru z opožděnou vazbou.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Tato funkce vrací vstupní řetězec.

Jestliže se funkce přeruší přes `\BreakFlag`:

- Jestliže je určen parametr `\InitString`, tento řetězec je vrácen
- Jestliže není určen parametr `\InitString`, je vrácen prázdný řetězec `""`.

Jestliže se funkce přeruší přes chybový handler `ERROR`, nebude vrácena žádná hodnota.

Argumenty

```
UIAlphaEntry ([\Header] [\Message]|[\MsgArray]
     [\Wrap][\Icon][\InitString] [\MaxTime] [\DIBreak] [\DIPassive]
     [\DOBreak] [\DOPassive] [\BreakFlag])
```

`[\Header]`

Datový typ: `string`

Text hlavičky, který bude zapsán v horní části okénka zprávy. Max. 40 znaků.

`[\Message]`

Datový typ: `string`

Jedna textová řádka bude zapsána na displej. Max. 55 znaků.

`[\MsgArray]`

Message Array

Datový typ: `string`

Several text lines from an array to be written on the display.

Pouze jeden z parametrů `\Message` , nebo `\MsgArray` se může použít ve stejnou dobu.

Max. prostor je 9 řádek s 55 znaky.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Only one of the predefined icons of type icondata can be used. Viz *Předdefinovaná data na str 1385*.

Standardně žádná ikona.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.189 UIAlphaEntry - Uživatelský vstup
RobotWare-OS
Pokračování

[\InitString]

Datový typ: string

Počáteční řetězec, který bude zobrazen v boxu pro vkládání textů jako výchozí.

[\MaxTime]

Datový typ: num

Max množství času v sekundách, kdy vykonávání programu čeká. Jestliže během této doby není stisknuta klávesa OK, program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_MAXTIME může být použita pro testování, jestli uplynula max doba nebo nikoliv.

[\DIBreak]

Digital Input Break

Datový typ: signaldi

Digitální vstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto tlačítko OK před nastavením signálu na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_DIBREAK může být použita pro testování, jestli toto vzniklo nebo nikoliv.

[\DIPassive]

Digital Input Passive

Datový typ: switch

Tento přepínač potlačuje standardní chování při použití volitelného argumentu DIBreak. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný BreakFlag), když je signál DIBreak nastaven na 0 (nebo již je 0). Konstantu ERR_TP_DIBREAK je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

[\DOBreak]

Digital Output Break

Datový typ: signaldo

Digitální výstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto tlačítko OK před nastavením signálu na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_DOBREAK může být použita pro testování, jestli toto vzniklo nebo nikoliv.

[\DOPassive]

Digital Output Passive

Datový typ: switch

Tento přepínač potlačuje standardní chování při použití volitelného argumentu DOBreak. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný BreakFlag), když je signál DOBreak nastaven na 0 (nebo již je 0). Konstantu ERR_TP_DOBREAK je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

[\BreakFlag]

Datový typ: errnum

Proměnná (před použitím je nastavena systémem na 0), která bude držet chybový kód, jestliže je použit\MaxTime, \DIBreak, **nebo**\DOBreak**. Konstanty ERR_TP_MAXTIME,** ERR_TP_MAXTIME, ERR_TP_DIBREAK, **a** ERR_TP_DOBREAK **mohou být použity pro volbu příčiny. Jestliže tato volitelná proměnná není použita, potom bude vykonán chybový handler.**

Vykonávání programu

Okénko pro zprávu s alpha padem, ikonou, hlavičkou, řádkami pro zprávu a init řetězcem se zobrazí podle naprogramovaných argumentů. Vykonávání programu čeká, až uživatel edituje nebo vytvoří nový řetězec a stiskne OK nebo až je okénko zpráv přerušeno vypršením času nebo činností signálu. Vstupní řetězec a důvod přerušení jsou převedeny zpět do programu.

Nové okénko pro zprávu na úrovni TRAP přebírá prioritu od okénka pro zprávu na základní úrovni.

Předdefinovaná data

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
```

Další příklady

Následující příklad názorně ukazuje funkciUIAlphaEntry.

Příklad 1

```
VAR errnum err_var;
VAR string answer;
VAR string logfile;
...
answer := UIAlphaEntry (\Header:= "Log file name:"
  \Message:= "Enter the name of the log file to create?"
  \Icon:=iconInfo
  \InitString:= "signal.log"
  \MaxTime:=60
  \DIBreak:=di5\BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer
    logfile:="signal.log";
  CASE 0:
    ! Operator answer
    logfile := answer;
  DEFAULT:
    ! No such case defined
ENDTEST
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.189 UIAlphaEntry - Uživatelský vstup
RobotWare-OS
Pokračování

Okénko pro zprávu je zobrazeno a operátor může vložit řetězec a stisknout OK. Okénko zpráv může být také přerušeno s vypršením času nebo digitálním vstupním signálem. V programu je možné vyhledat důvod a vykonat příslušnou činnost.

Řešení chyb

Jestliže parametr `\BreakFlag` není použit, tyto situace mohou být potom ošetřeny chybovým handlerem:

Jestliže se objevilo vypršení času (parametr `\MaxTime`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_MAXTIME` a vykonávání pokračuje v chybovém handleru.

Jestliže je nastaven digitální vstup (parametr `DIBreak`) před vstupem operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DIBREAK` a vykonávání pokračuje v chybovém handleru.

Jestliže digitální výstup je nastaven (parametr `DOBreak`)) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DOBREAK` a vykonávání pokračuje v chybovém handleru.

Tato situace může být potom ošetřena chybovým handlerem:

Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`, systémová proměnná `ERRNO` je nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.

Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná `ERRNO` je nastavena na `ERR_TP_NO_CLIENT` a vykonávání pokračuje v chybovém handleru.

Omezení

Vyhněte se používání příliš malých hodnot pro parametr vypršení času `\MaxTime`, když je často vykonáván `UIAlphaEntry`, např. ve smyčce. Výsledkem může být nepředvídané chování činnosti systému, jako je zpomalení odezvy FlexPendantu.

Syntaxe

```
UIAlphaEntry '('
  ['\' Header ':=' <expression (IN) of string>]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':='<array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' InitString ':='<expression (IN) of string>]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':='<variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>] ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Data ikony displeje	*icondata - Data ikony displeje na str 1512*
Základní typ boxu zpráv uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Pokročilý typ boxu zpráv uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*
Numerický vstup uživatelské interakce	*UINumEntry - Uživatelský číselný vstup na str 1417*
Numerické ladění uživatelské interakce	*UINumTune - Ladění uživatelských čísel na str 1423*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Systém připojen k FlexPendantu atd.	*UIClientExist - Existence uživatelského klienta na str 1388*
Volání procedury s opožděnou vazbou	*Technická referenční příručka - Přehled RAPID*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.190 UIClientExist - Existence uživatelského klienta

Použití

`UIClientExist` (*User Interaction Client Exist*) se používá ke kontrole, jestli některé uživatelské zařízení, jako je Flex Pendant, je připojeno k řadiči.

Základní příklady

Následující příklad názorně ukazuje funkci`UIClientExist`.

Příklad 1

```
IF UIClientExist() THEN
  ! Possible to get answer from the operator
  ! The TPReadFK and UIMsgBox ... can be used
ELSE
  ! Not possible to communicate with any operator
ENDIF
```

Je proveden test, jestli je možné získat nějakou odpověď od operátora systému.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Vrací `TRUE`, jestliže FlexPendant je připojen k systému, jinak `FALSE`.

Omezení

`UIClientExist` vrací `TRUE` až 16 sekund. Potom je FlexPendant odstraněn. Po této době `UIClientExist` vrací `FALSE` (tj. když je zjištěno ztracení připojení sítě od FlexPendantu). Stejné omezení platí, když je FlexPendant znovu připojen.

Syntaxe

```
UIClientExist '(' ')'
```

Funkce s vrácenou hodnotou typu `bool`.

Související informace

Pro informace o	Viz
Základní typ boxu zpráv uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Pokročilý typ boxu zpráv uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*
Numerický vstup uživatelské interakce	*UINumEntry - Uživatelský číselný vstup na str 1417*
Numerické ladění uživatelské interakce	*UINumTune - Ladění uživatelských čísel na str 1423*
Alfa vstup uživatelské interakce	*UIAlphaEntry - Uživatelský vstup na str 1382*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.191 UIDnumEntry - Uživatelský vstup čísel

Použití

`UIDnumEntry` (*User Interaction Number Entry*) se používá pro umožnění operátorovi vložit numerickou hodnotu z dostupného uživatelského zařízení, jako je FlexPendant. Zpráva je napsána pro operátora, který odpoví numerickou hodnotou. Numerická hodnota je potom zkontrolována, schválena a přenesena zpět do programu.

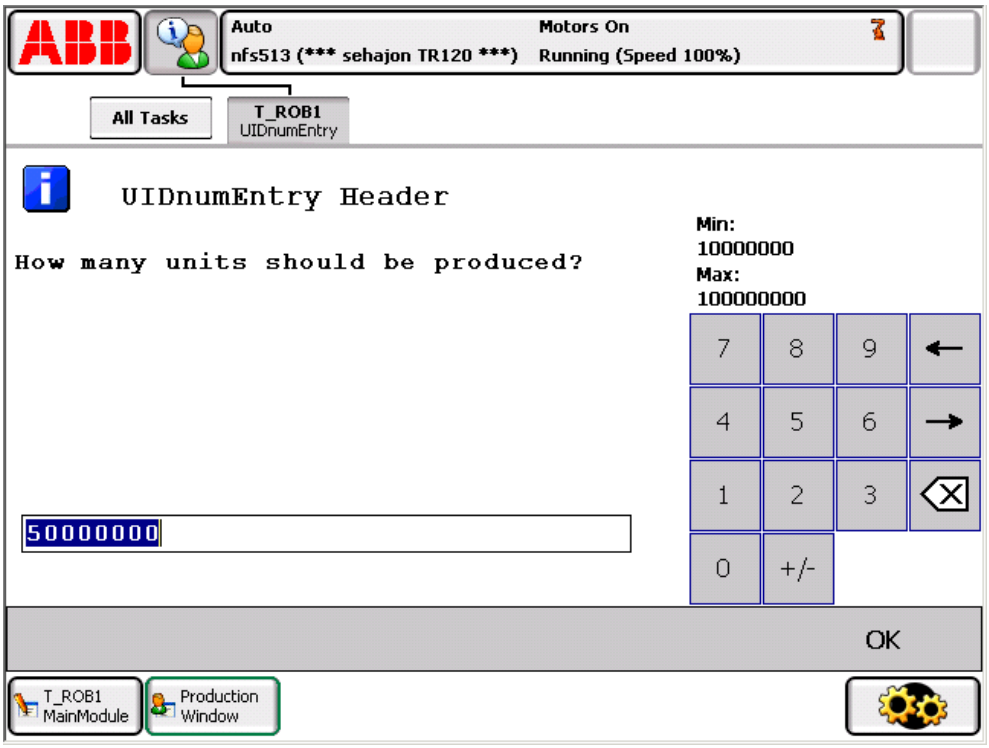
Základní příklady

Následující příklad názorně ukazuje funkci`UIDnumEntry`.

Viz také *Další příklady na str 1392*.

Příklad 1

```
VAR dnum answer;
...
answer := UIDnumEntry(
  \Header:="UIDnumEntry Header"
  \Message:="How many units should be produced?"
  \Icon:=iconInfo
  \InitValue:=50000000
  \MinValue:=10000000
  \MaxValue:=100000000
  \AsInteger);
```



xx0900001064

Nahoře, okénko pro numerickou zprávu s ikonou, hlavičkou, zprávou, init-, max- a minvalue zapsanými na displej FlexPendantu. Okénko zpráv kontroluje, aby

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

operátor zvolil celé číslo v rámci rozpětí hodnoty. Vykonávání programu čeká na stisknutí OK a potom je vrácena zvolená numerická hodnota.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Tato funkce vrací vstupní numerickou hodnotu.

Jestliže se funkce přeruší přes `\BreakFlag`:

- Jestliže je určen parametr `\InitValue`, tato hodnota je vrácena
- Jestliže není určen parametr `\InitValue`, je vrácena hodnota 0.

Jestliže se funkce přeruší přes chybový handler `ERROR`, nebude vrácena žádná hodnota.

Argumenty

```
UIDnumEntry ( [\Header] [\Message] | [\MsgArray]
     [\Wrap][\Icon][\InitValue] [\MinValue] [\MaxValue]
     [\AsInteger][\MaxTime] [\DIBreak] [\DIPassive] [\DOBreak]
     [\DOPassive] \BreakFlag])
```

`[\Header]`

Datový typ: `string`

Text hlavičky, který bude zapsán v horní části okénka zprávy. Max. 40 znaků.

`[\Message]`

Datový typ: `string`

Jedna textová řádka bude zapsána na displej. Max. 40 znaků.

`[\MsgArray]`

Message Array

Datový typ: `string`

Several text lines from an array to be written on the display.

Pouze jeden z parametrů `\Message`, nebo `\MsgArray` se může použít ve stejnou dobu.

Max. prostor je 9 řádek se 40 znaky na řádku.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 1392*.

Standardně žádná ikona.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

[\InitValue]

Datový typ: dnum

Počáteční hodnota, která je zobrazena ve vstupním okénku.

[\MinValue]

Datový typ: dnum

Minimální hodnota pro vracenou hodnotu.

[\MaxValue]

Datový typ: dnum

Maximální hodnota pro vracenou hodnotu.

[\AsInteger]

Datový typ: switch

Eliminuje desetinnou tečku z numerické klávesnice, aby bylo zajištěno, že vracená hodnota bude celé číslo.

[\MaxTime]

Datový typ: num

Max množství času v sekundách, kdy vykonávání programu čeká. Jestliže během této doby není stisknuta klávesa OK, program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_MAXTIME může být použita pro testování, jestli uplynula max doba nebo nikoliv.

[\DIBreak]

Digital Input Break

Datový typ: signaldi

Digitální vstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto tlačítko OK před nastavením signálu na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_DIBREAK může být použita pro testování, jestli toto vzniklo nebo nikoliv.

[\DIPassive]

Digital Input Passive

Datový typ: switch

Tento přepínač potlačuje standardní chování při použití volitelného argumentu DIBreak. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný BreakFlag), když je signál DIBreak nastaven na 0 (nebo již je 0). Konstantu ERR_TP_DIBREAK je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

[\DOBreak]

Digital Output Break

Datový typ: signaldo

Digitální výstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto tlačítko OK před nastavením signálu na 1 (nebo již je 1), program

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.191 UIDnumEntry - Uživatelský vstup čísel
RobotWare - OS
Pokračování

pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DOBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DOPassive]`

Digital Output Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DOBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DOBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DOBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\BreakFlag]`

Datový typ: `errnum`

Proměnná (před použitím je nastavena systémem na 0), která bude držet chybový kód, jestliže je použit`\MaxTime`, `\DIBreak`, nebo`\DOBreak`. Konstanty `ERR_TP_MAXTIME`, `ERR_TP_DIBREAK` a `ERR_TP_DOBREAK` mohou být použity pro volbu příčiny. Jestliže tato volitelná proměnná není použita, potom bude vykonán chybový handler.

Vykonávání programu

Numerické okénko pro zprávu s numerickou klávesnicí, ikonou, hlavičkou, řádkami pro zprávu a init-, max- a minvalue se zobrazí podle naprogramovaných argumentů. Vykonávání programu čeká, až uživatel vloží schválenou numerickou hodnotu a stiskne OK nebo až je okénko zpráv přerušeno vypršením času nebo činností signálu. Vstupní numerická hodnota a důvod přerušení jsou přeneseny zpět do programu.

Nové okénko pro zprávu na úrovni TRAP přebírá prioritu od okénka pro zprávu na základní úrovni.

Předdefinovaná data

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
```

Další příklady

Následující příklad názorně ukazuje funkci`UIDnumEntry`.

Příklad 1

```
VAR errnum err_var;
VAR dnum answer;
VAR dnum distance;
...
answer := UIDnumEntry (\Header:= "BWD move on path"
  \Message:="Enter the path overlap?" \Icon:=iconInfo
  \InitValue:=5 \MinValue:=0 \MaxValue:=10
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  \MaxTime:=60 \DIBreak:=di5 \BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer distance := 5;
  CASE 0
    ! Operator answer
    distance := answer;
  DEFAULT:
    ! No such case defined
ENDTEST
```

Okénko pro zprávu je zobrazeno a operátor může vložit numerickou hodnotu a stisknout OK. Okénko zpráv může být také přerušeno s vypršením času nebo digitálním vstupním signálem. V programu je možné vyhledat důvod a vykonat příslušnou činnost.

Řešení chyb

Jestliže parametr `\BreakFlag` není použit, tyto situace mohou být potom ošetřeny chybovým handlerem:

- Jestliže se objevilo vypršení času (parametr `\MaxTime`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_MAXTIME` a vykonávání pokračuje v chybovém handleru.
- Jestliže je nastaven digitální vstup (parametr `\DIBreak`) před vstupem operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DIBREAK` a vykonávání pokračuje v chybovém handleru.
- Jestliže digitální výstup nastaven (parametr `\DOBreak`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DOBREAK` a vykonávání pokračuje v chybovém handleru.

Tato situace může být potom ošetřena chybovým handlerem:

- Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`, `ERRNO` je nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.
- Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná `ERRNO` je nastavena na `ERR_TP_NO_CLIENT` a vykonávání pokračuje v chybovém handleru.
- Počáteční hodnota (parametr `\InitValue`) není určena v rámci rozsahu min a max hodnoty (parametry `\MinValue` a `\MaxValue`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UI_INITVALUE` a vykonávání pokračuje v chybovém handleru.
- Jestliže minimální hodnota (parametr `\MinValue`) je větší než max hodnota (parametr `\MaxValue`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UI_MAXMIN` a vykonávání pokračuje v chybovém handleru.
- Jestliže počáteční hodnota (parametr `\InitValue`) není celé číslo, jak je určeno v parametru `\AsInteger`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UI_NOTINT` a vykonávání pokračuje v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.191 UIDnumEntry - Uživatelský vstup čísel

RobotWare - OS

Pokračování

Omezení

Vyhněte se používání příliš malých hodnot pro parametr vypršení času `\MaxTime`, když je často vykonáván `UIDnumEntry`, např. ve smyčce. Výsledkem může být nepředvídané chování činnosti systému, jako je zpomalení odezvy FlexPendantu.

Syntaxe

```
UIDnumEntry '('
  ['\' Header ':=' <expression (IN) of string>]
  [Message ':=' <expression (IN) of string> ]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' InitValue ':=' <expression (IN) of dnum>]
  ['\' MinValue ':=' <expression (IN) of dnum>]
  ['\' MaxValue ':=' <expression (IN) of dnum>]
  ['\' AsInteger]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>] ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Data ikony displeje	*icondata - Data ikony displeje na str 1512*
Základní typ boxu zpráv uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Pokročilý typ boxu zpráv uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*
Numerický vstup uživatelské interakce	*UINumEntry - Uživatelský číselný vstup na str 1417*
Numerické ladění uživatelské interakce	*UIDnumTune - Ladění uživatelských čísel na str 1395*
Numerické ladění uživatelské interakce	*UINumTune - Ladění uživatelských čísel na str 1423*
Alfa vstup uživatelské interakce	*UIAlphaEntry - Uživatelský vstup na str 1382*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Systém připojen k FlexPendantu atd.	*UIClientExist - Existence uživatelského klienta na str 1388*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.192 UIDnumTune - Ladění uživatelských čísel

Použití

`UIDnumTune` (*User Interaction Number Tune*) se používá pro umožnění operátorovi ladit numerickou hodnotu z dostupného uživatelského zařízení, jako je FlexPendant. Zpráva je napsána pro operátora, který ladí numerickou hodnotu. Naladěná numerická hodnota je potom zkontrolována, schválena a přenesena zpět do programu.

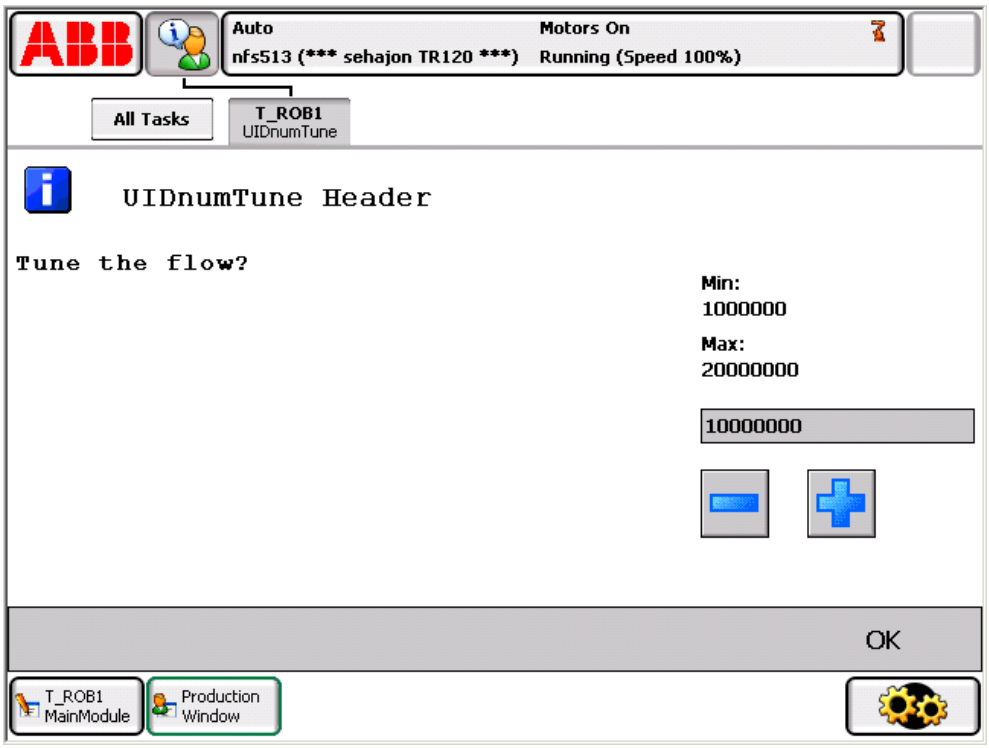
Základní příklady

Následující příklad názorně ukazuje funkci`UIDnumTune`.

Viz také *Další příklady na str 1398*.

Příklad 1

```
VAR dnum flow;
...
flow := UIDnumTune(
  \Header:="UIDnumTune Header"
  \Message:="Tune the flow?"
  \Icon:=iconInfo,
  10000000,
  1000000
  \MinValue:=1000000
  \MaxValue:=20000000);
```



xx0900001063

Nahoře, okénko pro zprávy numerického ladění s ikonou, hlavičkou, zprávou, init-, přírůstkovou, max- a minvalue zapsanými na displej FlexPendantu. Okénko zpráv

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.192 UIDnumTune - Ladění uživatelských čísel

RobotWare - OS

Pokračování

kontroluje, že operátor ladí hodnotu `flow` s krokem 1000000 od init hodnoty 10000000 a drží se v rozsahu hodnoty 1000000-20000000. Vykonávání programu čeká na stisknutí OK a potom je vrácena zvolená numerická hodnota a uložena do proměnné `flow`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `dnum`

Tato funkce vrací naladěnou numerickou hodnotu.

Jestliže se funkce přeruší přes `\BreakFlag`, je vrácena určená `InitValue`.

Jestliže se funkce přeruší přes chybový handler `ERROR`, nebude vrácena žádná hodnota.

Argumenty

```
UIDnumTune ( [\Header] [\Message] | [\MsgArray] [\Wrap]
     [\Icon]InitValue Increment [\MinValue] [\MaxValue]
     [\MaxTime][\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive]
     [\BreakFlag] )
```

`[\Header]`

Datový typ: `string`

Text hlavičky, který bude zapsán v horní části okénka zprávy. Max. 40 znaků.

`[\Message]`

Datový typ: `string`

Jedna textová řádka bude zapsána na displej. Max. 40 znaků.

`[\MsgArray]`

Message Array

Datový typ: `string`

Several text lines from an array to be written on the display.

Pouze jeden z parametrů `\Message`, nebo `\MsgArray` se může použít ve stejnou dobu.

Max. prostor je 11 řádek se 40 znaky na řádku.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Only one of the predefined icons of type icondata can be used. Viz *Předdefinovaná data na str 1398*.

Standardně žádná ikona.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`InitValue`

Initial Value

Datový typ: `dnum`

Počáteční hodnota, která je zobrazena ve vstupním okénku.

`Increment`

Datový typ: `dnum`

Tento parametr určuje, jak hodně by se hodnota měla změnit, když je stisknuto tlačítko plus nebo minus.

`[\MinValue]`

Datový typ: `dnum`

Minimální hodnota pro vracenou hodnotu.

`[\MaxValue]`

Datový typ: `dnum`

Maximální hodnota pro vracenou hodnotu.

`[\MaxTime]`

Datový typ: `num`

Max množství času v sekundách, kdy vykonávání programu čeká. Jestliže během této doby není stisknuta klávesa OK, program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_MAXTIME` může být použita pro testování, jestli uplynula max doba nebo nikoliv.

`[\DIBreak]`

Digital Input Break

Datový typ: `signaldi`

Digitální vstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto tlačítko OK před nastavením signálu na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DIBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DIPassive]`

Digital Input Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DIBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DIBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DIBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\DOBreak]`

Digital Output Break

Datový typ: `signaldo`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.192 UIDnumTune - Ladění uživatelských čísel
RobotWare - OS
Pokračování

Digitální výstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto tlačítko OK před nastavením signálu na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DOBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DOPassive]`

Digital Output Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DOBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DOBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DOBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\BreakFlag]`

Datový typ: `errnum`

Proměnná (před použitím je nastavena systémem na 0), která bude držet chybový kód, jestliže je použit`\MaxTime`, `\DIBreak`, nebo`\DOBreak`. Konstanty `ERR_TP_MAXTIME`, `ERR_TP_DIBREAK` a `ERR_TP_DOBREAK` mohou být použity pro volbu příčiny. Jestliže tato volitelná proměnná není použita, potom bude vykonán chybový handler.

Vykonávání programu

Okénko zpráv numerického ladění s ladicími tlačítky +/-, ikonou, hlavičkou, řádkami pro zprávu a init-, přírůstkovou, max- a minvalue se zobrazí podle naprogramovaných argumentů. Vykonávání programu čeká, až uživatel vloží schválenou numerickou hodnotu a stiskne OK nebo až je okénko zpráv přerušeno vypršením času nebo činností signálu. Vstupní numerická hodnota a důvod přerušení jsou přeneseny zpět do programu.

Nové okénko pro zprávu na úrovni `TRAP` přebírá prioritu od okénka pro zprávu na základní úrovni.

Předdefinovaná data

```
!Icons:
CONST icondata iconNone := 0;
CONST icondata iconInfo := 1;
CONST icondata iconWarning := 2;
CONST icondata iconError := 3;
```

Další příklady

Následující příklad názorně ukazuje funkci`UIDnumTune`.

Příklad 1

```
VAR errnum err_var;
VAR dnum tune_answer;
VAR dnum distance;
...
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
tune_answer := UIDnumTune (\Header:=" BWD move on path"
  \Message:="Enter the path overlap?" \Icon:=iconInfo,
  5, 1 \MinValue:=0 \MaxValue:=10
  \MaxTime:=60 \DIBreak:=di5 \BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer
    distance := 5;
  CASE 0:
    ! Operator answer
    distance := tune_answer;
  DEFAULT:
    ! No such case defined
ENDTEST
```

Okénko pro zprávu ladění je zobrazeno a operátor může ladit numerickou hodnotu a stisknout OK. Okénko zpráv může být také přerušeno s vypršením času nebo digitálním vstupním signálem. V programu je možné vyhledat důvod a vykonat příslušnou činnost.

Řešení chyb

Jestliže parametr `\BreakFlag` není použit, tyto situace mohou být ošetřeny chybovým handlerem:

- Jestliže se objevilo vypršení času (parametr `\MaxTime`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_MAXTIME` a vykonávání pokračuje v chybovém handleru.
- Jestliže je nastaven digitální vstup (parametr `\DIBreak`) před vstupem operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DIBREAK` a vykonávání pokračuje v chybovém handleru.
- Jestliže digitální výstup je nastaven (parametr `\DOBreak`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DOBREAK` a vykonávání pokračuje v chybovém handleru.

Tato situace může být potom ošetřena chybovým handlerem:

- Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`, `ERRNO` je nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.
- Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná `ERRNO` je nastavena na `ERR_TP_NO_CLIENT` a vykonávání pokračuje v chybovém handleru.
- Počáteční hodnota (parametr `\InitValue`) není určena v rámci rozsahu min a max hodnoty (parametry `\MinValue` a `\MaxValue`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UI_INITVALUE` a vykonávání pokračuje v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.192 UIDnumTune - Ladění uživatelských čísel

RobotWare - OS

Pokračování

- Jestliže minimální hodnota (parametr `\MinValue`) je větší než max hodnota (parametr `\MaxValue`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UI_MAXMIN` a vykonávání pokračuje v chybovém handleru.

Omezení

Vyhněte se používání příliš malých hodnot pro parametr vypršení času `\MaxTime`, když je často vykonáván `UIDnumTune`, např. ve smyčce. Výsledkem může být nepředvídané chování činnosti systému, jako je zpomalení odezvy FlexPendantu.

Syntaxe

```
UIDnumTune '('
  ['\' Header ':=' <expression (IN) of string>]
  ['\' Message ':=' <expression (IN) of string> ]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>] ',']
  [InitValue ':=' ] <expression (IN) of dnum> ','
  [Increment ':=' ] <expression (IN) of dnum>
  ['\' MinValue ':=' <expression (IN) of dnum>]
  ['\' MaxValue ':=' <expression(IN) of dnum>]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>] ')'
```

Funkce s vrácenou hodnotou datového typu `dnum`.

Související informace

Pro informace o	Viz
Data ikony displeje	*icondata - Data ikony displeje na str 1512*
Základní typ boxu zpráv uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Pokročilý typ boxu zpráv uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*
Numerický vstup uživatelské interakce	*UIDnumEntry - Uživatelský vstup čísel na str 1389*
Numerický vstup uživatelské interakce	*UINumEntry - Uživatelský číselný vstup na str 1417*
Numerické ladění uživatelské interakce	*UINumTune - Ladění uživatelských čísel na str 1423*
Alfa vstup uživatelské interakce	*UIAlphaEntry - Uživatelský vstup na str 1382*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Systém připojen k FlexPendantu atd.	*UIClientExist - Existence uživatelského klienta na str 1388*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.193 UIListView - Náhled uživatelského seznamu

Použití

`UIListView` (*User Interaction List View*) se používá k definování nabídkových seznamů s textem a volitelnými ikonami na dostupném uživatelském zařízení, jako je FlexPendant. Nabídka má dva různé styly, jeden s tlačítky vyhodnocení a jeden, který stále reaguje na volbu uživatele.

Základní příklady

Následující příklad názorně ukazuje funkci`UIListView`.

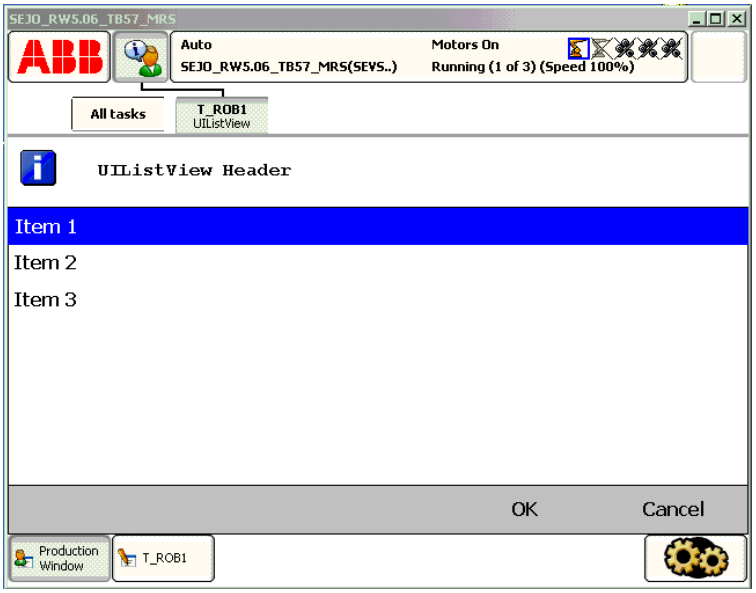
Viz také *Další příklady na str 1407*.

Příklad 1

```
CONST listitem list{3} := [ ["","Item 1"], ["","Item 2"],
     ["","Item3"] ];
VAR num list_item;
VAR btnres button_answer;
...
list_item := UIListView (
  \Result:=button_answer
  \Header:="UIListView Header",
  list
  \Buttons:=btnOKCancel
  \Icon:=iconInfo
  \DefaultIndex:=1);
IF button_answer = resOK THEN
  IF list_item = 1 THEN
    ! Do item1
  ELSEIF list_item = 2 THEN
    ! Do item 2
  ELSE
    ! Do item3
  ENDIF
ELSE
  ! User has select Cancel
ENDIF
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.



xx0500002416

Shora uvedený nabídkový seznam s ikonou, hlavičkou, nabídkou `Item 1 ... Item 3` a tlačítky jsou zapsány na displej FlexPendantu. Vykonávání programu čeká na stisknutí OK nebo Zrušit. Volba v seznamu a stisknuté tlačítko jsou přeneseny do programu.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Tato funkce vrací volbu uživatele v nabídkovém seznamu odpovídající indexu v poli určeném v parametru `ListItems`.

Jestliže se funkce přeruší přes `\BreakFlag`:

- Jestliže je určen parametr`\DefaultIndex`, tento index je vrácen
- Jestliže není určen parametr`\DefaultIndex`, je vrácena hodnota 0

Jestliže se funkce přeruší přes chybový handler `ERROR`, nebude vrácena žádná hodnota.

Argumenty

```
UIListView ( [\Result] [\Header] ListItems [\Buttons] |
     [\BtnArray][\Icon] [\DefaultIndex ] [\MaxTime] [\DIBreak]
     [\DIPassive][\DOBreak] [\DOPassive] [\BreakFlag])
```

`[\Result]`

Datový typ: `btnres`

Numerická hodnota tlačítka, které je zvoleno z okénka nabídkového seznamu.

Jestliže je použit argument `\Buttons`, předdefinované symbolické konstanty typu `btnres` jsou vráceny. Jestliže je použit argument `\BtnArray`, je vrácen odpovídající index pole.

Argument `\Result` nastavený na resUnkwn rovný 0, jestliže se jedná o jednu z následujících podmínek:

- není použit žádný z parametrů `\Buttons` nebo `\BtnArray`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.193 UIListView - Náhled uživatelského seznamu
RobotWare - OS
Pokračování

- je použit argument `\Buttons:=btnNone`
- jestliže se funkce přeruší přes`\BreakFlag` nebo chybový handler `ERROR`

Viz část *Předdefinovaná data na str 1406*.

`[\Header]`

Datový typ: `string`

Text hlavičky, který bude zapsán v horní části okénka nabídkového seznamu. Max. 40 znaků.

`ListItems`

Datový typ: `listitem`

Pole s jednou nebo více položkami nabídkového seznamu, které bude zobrazeno a skládá se z:

Komponent `image` typu `string`:

Jméno ikonového obrázku, který by měl být použit. Chcete-li poslat své vlastní obrázky, musí být umístěny do adresáře `HOME:` v aktivním systému nebo přímo do aktivního systému.

Doporučuje se umístit soubory do adresáře `HOME:`, aby byly uloženy při provádění zálohování nebo obnovy.

Vyžaduje se **Restart** a potom FlexPendant načte obrázky.

Požadavkem na systém je, aby byl použit doplněk RobotWare *FlexPendant Interface*.

Obrázek, který bude zobrazen, může mít šířku a výšku 28 pixelů. Jestliže je obrázek větší, bude upraven na velikost pouze 28 * 28 pixelů.

Není možné určovat přesnou hodnotu velikosti obrázků nebo počet obrázků, které mohou být načteny do FlexPendantu. Závisí to na velikosti ostatních souborů načtených do FlexPendantu. Vykonávání programu bude pouze pokračovat, jestliže je použit obrázek, který nebyl načten do FlexPendantu.

Použijte prázdný řetězec "" nebo stEmpty, jestliže není žádná ikona k zobrazení.

Komponent `text` typu `string`:

- Text pro řádku nabídky k zobrazení.
- Max. 75 znaků pro každou položku nabídkového seznamu.

`[\Buttons]`

Datový typ: `buttondata`

Definuje tlačítka, která budou zobrazena. Může se použít pouze jedna z předdefinovaných kombinací tlačítek typu buttondata. Viz *Předdefinovaná data na str 1406*.

`[\BtnArray]`

Button Array

Datový typ: `string`

Vlastní definice tlačítek uložená do pole řetězců. Tato funkce vrací index pole, když je zvolen odpovídající řetězec.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pouze jeden z parametrů `\Buttons` nebo `\BtnArray` může být použit ve stejnou dobu. Jestliže není použit žádný z parametrů `\Buttons` nebo `\BtnArray` nebo argument `\Buttons:=btnNone`, potom nabídkový seznam stále reaguje na volbu uživatele.

Max. 5 tlačítek, každé se 42 znaky.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`.

Standardně žádná ikona. Viz *Předdefinovaná data na str 1406*.

`[\DefaultIndex]`

Datový typ: `num`

Výchozí volba uživatele v nabídkovém seznamu odpovídající indexu v poli určeném v parametru `ListItems`.

`[\MaxTime]`

Datový typ: `num`

Max množství času v sekundách, kdy vykonávání programu čeká. Jestliže během této doby není zvoleno žádné tlačítko a není provedena žádná volba, program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_MAXTIME` může být použita pro testování, jestli uplynula max doba nebo nikoliv.

`[\DIBreak]`

Digital Input Break

Datový typ: `signaldi`

Digitální vstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto žádné tlačítko nebo není provedena žádná volba předtím, než je signál nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DIBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DIPassive]`

Digital Input Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DIBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DIBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DIBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\DOBreak]()`

Digital Output Break

Datový typ: `signaldo`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Digitální výstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto žádné tlačítko nebo není provedena žádná volba předtím, než je signál nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DOBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DOPassive]`

Digital Output Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DOBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DOBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DOBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\BreakFlag]`

Datový typ: `errnum`

Proměnná, která bude držet chybový kód, jestliže je použit`\MaxTime`, `\DIBreak`, nebo`\DOBreak`. Konstanty `ERR_TP_MAXTIME`, `ERR_TP_DIBREAK` a `ERR_TP_DOBREAK` mohou být použity pro volbu příčiny. Jestliže tato volitelná proměnná není použita, potom bude vykonán chybový handler.

Vykonávání programu

Nabídkový seznam s ikonou, hlavičkou, položkami seznamu a výchozí položkou se zobrazí podle naprogramovaných argumentů. Vykonávání programu čeká, až uživatel provede volbu nebo až je nabídkový seznam přerušen vypršením času nebo činností signálu. Zvolený nabídkový seznam a důvod přerušení jsou převedeny zpět do programu.

Nový nabídkový seznam na úrovni TRAP přebírá prioritu od nabídkového seznamu na základní úrovni.

Předdefinovaná data

```
!Icons:
CONST icondata iconNone := 0;
CONST icondata iconInfo := 1;
CONST icondata iconWarning := 2;
CONST icondata iconError := 3;
!Buttons:
CONST buttondata btnNone := -1;
CONST buttondata btnOK := 0;
CONST buttondata btnAbrtRtryIgn := 1;
CONST buttondata btnOKCancel := 2;
CONST buttondata btnRetryCancel := 3;
CONST buttondata btnYesNo := 4;
CONST buttondata btnYesNoCancel := 5;
!Results:
CONST btnres resUnkwn := 0;
CONST btnres resOK := 1;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
CONST btnres resAbort := 2;
CONST btnres resRetry := 3;
CONST btnres resIgnore := 4;
CONST btnres resCancel := 5;
CONST btnres resYes := 6;
CONST btnres resNo := 7;
```

Další příklady

Následující příklad názorně ukazuje funkci`UIListView`.

Příklad 1

```
CONST listitem list{2} := [ ["","Calibrate tool1"], ["","Calibrate
     tool2"] ];
VAR num list_item;
VAR errnum err_var;
...
list_item := UIListView
  ( \Header:="Select tool ?",
  list \Icon:=iconInfo
  \MaxTime:=60
  \DIBreak:=di5
  \BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer
  CASE 0:
    ! Operator answer
    IF list_item =1 THEN
      ! Calibrate tool1
    ELSEIF list_item=2 THEN
      ! Calibrate tool2
    ENDIF
  DEFAULT:
    ! Not such case defined
ENDTEST
```

Okénko pro zprávu je zobrazeno a operátor může zvolit položku v seznamu. Okénko zpráv může být také přerušeno s vypršením času nebo digitálním vstupním signálem. V programu je možné vyhledat důvod a vykonat příslušnou činnost.

Řešení chyb

Jestliže parametr `\BreakFlag` není použit, tyto situace mohou být potom ošetřeny chybovým handlerem:

- Jestliže se objevilo vypršení času (parametr `\MaxTime`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_MAXTIME` a vykonávání pokračuje v chybovém handleru.
- Jestliže je nastaven digitální vstup (parametr `\DIBreak`) před vstupem operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DIBREAK` a vykonávání pokračuje v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

- Jestliže digitální výstup je nastaven (parametr \DOBreak) před vstupem od operátora, systémová proměnná ERRNO je nastavena na ERR_TP_DOBREAK a vykonávání pokračuje v chybovém handleru.

Tato situace může být potom ošetřena chybovým handlerem:

- Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO, ERRNO je nastavena na ERR_NO_ALIASIO_DEF a vykonávání pokračuje v chybovém handleru.
- Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná ERRNO je nastavena na ERR_TP_NO_CLIENT a vykonávání pokračuje v chybovém handleru.

Omezení

Vyhněte se používání příliš malých hodnot pro parametr vypršení času \MaxTime, když je často vykonáván UIListView, např. ve smyčce. Výsledkem může být nepředvídané chování činnosti systému, jako je zpomalení odezvy FlexPendantu.

Syntaxe

```
UIListView '('
  [['\' Result ':=' <var or pers (INOUT) of btnres>]
  ['\' Header ':=' <expression (IN) of string>] ',']
  [ListItems '='] <array {*} (IN) of listitem>
  ['\' Buttons ':=' <expression (IN) of buttondata>]
  | ['\' BtnArray ':=' <array {*} (IN) of string>]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' DefaultIndex ':=' <expression (IN) of num>]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>] ')'
```

Funkce s vrácenou hodnotou datového typu num.

Související informace

Pro informace o	Viz
Data ikony displeje	*icondata - Data ikony displeje na str 1512*
Data tlačítka	*buttondata - Data tlačítka na str 1444*
Výsledková data tlačítka	*btnres - Výsledková data tlačítka na str 1441*
Datová struktura nabídkového seznamu	*listitem - Vypsat datovou strukturu položek na str 1522*
Základní typ boxu zpráv uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Pokročilý typ boxu zpráv uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Numerický vstup uživatelské interakce	*UINumEntry - Uživatelský číselný vstup na str 1417*
Numerické ladění uživatelské interakce	*UINumTune - Ladění uživatelských čísel na str 1423*
Alfa vstup uživatelské interakce	*UIAlphaEntry - Uživatelský vstup na str 1382*
Systém připojen k FlexPendantu atd.	*UIClientExist - Existence uživatelského klienta na str 1388*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.194 UIMessageBox - Pokročilý typ uživatelského boxu zpráv

Použití

`UIMessageBox` (*User Interaction Message Box*) se používá ke komunikaci s uživatelem systému robotu na dostupném uživatelském zařízení, jako je FlexPendant. Zpráva je napsána k operátorovi, který odpoví volbou tlačítka. Uživatelská volba je potom přenesena zpět do programu.

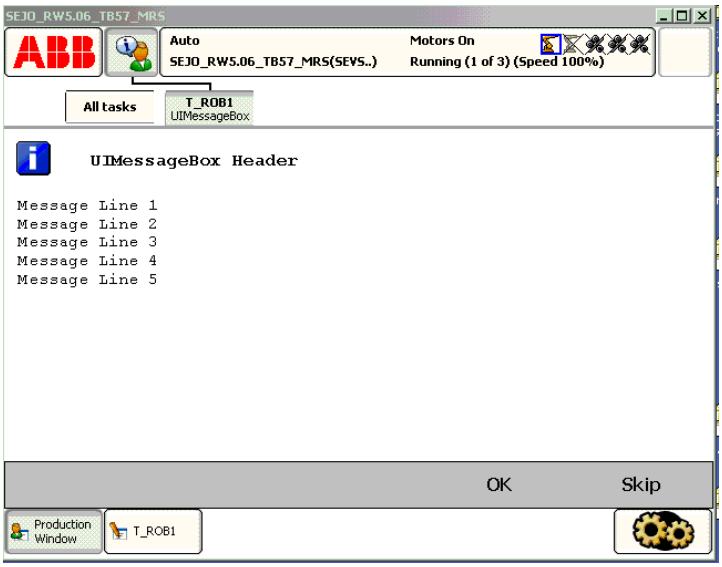
Základní příklady

Následující příklad názorně ukazuje funkci`UIMessageBox`.

Viz také *Další příklady na str 1415*.

Příklad 1

```
VAR btnres answer;
CONST string my_message{5}:= ["Message Line 1","Message Line 2",
     "Message Line 3","Message Line 4","Message Line 5"];
CONST string my_buttons{2}:=["OK","Skip"];
...
answer:= UIMessageBox (
  \Header:="UIMessageBox Header"
  \MsgArray:=my_message
  \BtnArray:=my_buttons
  \Icon:=iconInfo);
IF answer = 1 THEN
  ! Operator selection OK
ELSEIF answer = 2 THEN
  ! Operator selection Skip
ELSE
  ! No such case defined
ENDIF
```



xx0500002409

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Shora uvedené okénko pro zprávu na displeji FlexPendantu obsahuje ikonu, hlavičku, zprávu a uživatelsky definovaná tlačítka. Vykonávání programu čeká, až je stisknuto OK nebo Skip (Přeskočit). Jinými slovy, pro `answer` bude přiděleno 1 (OK) nebo 2 (Skip) podle toho, které tlačítko je stisknuto (odpovídající index pole).

POZNÁMKA

`Message Line 1` až `Message Line 5` jsou zobrazeny na samostatných řádkách 1 až 5 (přepínač `\Wrap` není použit).

Vratná hodnota (Vrátit hodnotu)

Datový typ: `btnres`

Numerická hodnota tlačítka, které je zvoleno z okénka pro zprávy.

Jestliže je použit argument `\Buttons`, jsou vráceny předdefinované symbolické konstanty typu `btnres`.

Jestliže je použit argument `\BtnArray`, je vrácen odpovídající index pole.

Jestliže se funkce přeruší přes `\BreakFlag` nebo když `\Buttons:=btnNone`:

- Jestliže je určen parametr`\DefaultBtn`, tento index je vrácen.
- Jestliže není určen parametr`\DefaultBtn`, je vrácen `resUnkwn` rovný 0.

Jestliže se funkce přeruší přes chybový handler `ERROR`, nebude vrácena žádná hodnota.

Argumenty

```
UIMessageBox ( [\Header] [\Message] | [\MsgArray] [\Wrap][\Buttons]
     | [\BtnArray] [\DefaultBtn] [\Icon][\Image] [\MaxTime]
     [\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive] [\BreakFlag])
```

`[\Header]`

Datový typ: `string`

Text hlavičky, který bude zapsán v horní části okénka zprávy. Max. 40 znaků.

`[\Message]`

Datový typ: `string`

Jedna textová řádka bude zapsána na displej. Max. 55 znaků.

`[\MsgArray]`

Message Array

Datový typ: `string`

Several text lines from an array to be written on the display.

Pouze jeden z parametrů `\Message`, nebo `\MsgArray` se může použít ve stejnou dobu.

Max. prostor je 11 řádek s 55 znaky na řádku.

`[\Wrap]`

Datový typ: `switch`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.194 UIMessageBox - Pokročilý typ uživatelského boxu zpráv
RobotWare - OS
Pokračování

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchými mezerami mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Buttons]`

Datový typ: `buttondata`

Defines the push buttons to be displayed. Only one of the predefined buttons combination of type buttondata can be used. Viz *Předdefinovaná data na str 1414*.

Systém zobrazuje standardně tlačítko OK.

`[\BtnArray]`

Button Array

Datový typ: `string`

Vlastní definice tlačítek uložená do pole řetězců. Tato funkce vrací index pole, když je zvolen odpovídající řetězec.

Pouze jeden z parametrů `\Buttons`, nebo `\BtnArray` se může použít ve stejnou dobu.

Max. 5 tlačítek, každé se 42 znaky.

`[\DefaultBtn]`

Default Button

Datový typ: `btnres`

Umožňuje určit hodnotu, která by měla být vrácena, jestliže okénko zpráv je přerušeno od `\MaxTime`, `\DIBreak`, or `\DOBreak`. Je možné určit předdefinovanou symbolickou konstantu typu `btnres` nebo jakoukoliv uživatelsky definovanou hodnotu. Viz *Předdefinovaná data na str 1414*.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 1414*.

Výchozí nastavení, žádná ikona.

`[\Image]`

Datový typ: `string`

Jméno obrázku, který by měl být použit. Chcete-li poslat své vlastní obrázky, musí být umístěny do adresáře `HOME:` v aktivním systému nebo přímo do aktivního systému.

Doporučuje se umístit soubory do adresáře `HOME:`, aby byly uloženy při provádění zálohování nebo obnovy.

Vyžaduje se **Restart** a potom FlexPendant načte obrázky.

Požadavkem na systém je, aby byl použit doplněk RobotWare *FlexPendant Interface*.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Obrázek, který má být zobrazen, může mít šířku 185 pixelů a výšku 300 pixelů. Jestliže je obrázek větší, pouze 185 * 300 pixelů obrázku bude vidět od horní levé části obrázku.

Není možné určovat přesnou hodnotu velikosti obrázků nebo počet obrázků, které mohou být načteny do FlexPendantu. Závisí to na velikosti ostatních souborů načtených do FlexPendantu. Vykonávání programu bude pouze pokračovat, jestliže je použit obrázek, který nebyl načten do FlexPendantu.

`[\MaxTime]`

Datový typ: `num`

Max množství času v sekundách, kdy vykonávání programu čeká. Jestliže během této doby není zvoleno žádné tlačítko, program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_MAXTIME` může být použita pro testování, jestli uplynula max doba nebo nikoliv.

`[\DIBreak]`

Digital Input Break

Datový typ: `signaldi`

Digitální vstupní signál, který může přerušit dialog operátora. Jestliže není zvoleno žádné tlačítko, když signál je nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DIBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DIPassive]`

Digital Input Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DIBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DIBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DIBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\DOBreak]`

Digital Output Break

Datový typ: `signaldo`

Digitální výstupní signál, který může přerušit dialog operátora. Jestliže není zvoleno žádné tlačítko, když signál je nastaven na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DOBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DOPassive]`

Digital Output Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DOBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.194 UIMessageBox - Pokročilý typ uživatelského boxu zpráv

RobotWare - OS

Pokračování

instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DOBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DOBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\BreakFlag]`

Datový typ: `errnum`

Proměnná (před použitím je nastavena systémem na 0), která bude držet chybový kód, jestliže je použit `\MaxTime`, , `\DIBreak`, nebo`\DOBreak`. Konstanty `ERR_TP_MAXTIME`, `ERR_TP_DIBREAK` a `ERR_TP_DOBREAK` mohou být použity pro volbu příčiny. Jestliže tato volitelná proměnná není použita, potom bude vykonán chybový handler.

Vykonávání programu

Okénko pro zprávu s ikonou, hlavičkou, řádkami pro zprávu, obrázkem a tlačítky se zobrazí podle naprogramovaných argumentů. Vykonávání programu čeká, až uživatel zvolí jedno tlačítko nebo až je okénko zpráv přerušeno vypršením času nebo činností signálu. Uživatelská volba a důvod přerušení jsou převedeny zpět do programu.

Nové okénko pro zprávu na úrovni TRAP přebírá prioritu od okénka pro zprávu na základní úrovni.

Předdefinovaná data

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
!Buttons:
  CONST buttondata btnNone := -1;
  CONST buttondata btnOK := 0;
  CONST buttondata btnAbrtRtryIgn := 1;
  CONST buttondata btnOKCancel := 2;
  CONST buttondata btnRetryCancel := 3;
  CONST buttondata btnYesNo := 4;
  CONST buttondata btnYesNoCancel := 5;
!Results:
  CONST btnres resUnkwn := 0;
  CONST btnres resOK := 1;
  CONST btnres resAbort := 2;
  CONST btnres resRetry := 3;
  CONST btnres resIgnore := 4;
  CONST btnres resCancel := 5;
  CONST btnres resYes := 6;
  CONST btnres resNo := 7;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Další příklady

Následující příklad názorně ukazuje funkci`UIMessageBox`.

Příklad 1

```
VAR errnum err_var;
VAR btnres answer;
...
answer := UIMessageBox (\Header:= "Cycle step 3"
  \Message:="Continue with the calibration ?" \Buttons:=btnOKCancel
  \DefaultBtn:=resCancel \Icon:=iconInfo \MaxTime:=60 \DIBreak:=di5
  \BreakFlag:=err_var);
IF answer = resOK THEN
  ! OK from the operator
ELSE
  ! Cancel from the operator or operation break
  TEST err_var
    CASE ERR_TP_MAXTIME:
      ! Time out
    CASE ERR_TP_DIBREAK:
      ! Input signal break
    DEFAULT:
      ! Not such case defined
  ENDTEST
ENDIF
```

Okénko pro zprávu je zobrazeno a operátor může odpovědět OK nebo Cancel. Okénko zpráv může být také přerušeno s vypršením času nebo digitálním vstupním signálem. V programu je možné vyhledat důvod.

Řešení chyb

Jestliže parametr `\BreakFlag` není použit, tyto situace mohou být potom ošetřeny chybovým handlerem:

- Jestliže se objevilo vypršení času (parametr `\MaxTime`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_MAXTIME` a vykonávání pokračuje v chybovém handleru.
- Jestliže je nastaven digitální vstup (parametr `\DIBreak`) před vstupem operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DIBREAK` a vykonávání pokračuje v chybovém handleru.
- Jestliže digitální výstup je nastaven (parametr `\DOBreak`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DOBREAK` a vykonávání pokračuje v chybovém handleru.

Tato situace může být potom ošetřena chybovým handlerem:

- Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`, `ERRNO` je nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.194 UIMessageBox - Pokročilý typ uživatelského boxu zpráv

RobotWare - OS

Pokračování

- Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná ERRNO je nastavena na ERR_TP_NO_CLIENT a vykonávání pokračuje v chybovém handleru.

Omezení

Vyhněte se používání příliš malých hodnot pro parametr vypršení času \MaxTime, když je často vykonáván UIMessageBox, např. ve smyčce. Výsledkem může být nepředvídané chování činnosti systému, jako je zpomalení odezvy FlexPendantu.

Syntaxe

```
UIMessageBox '('
  ['\' Header ':=' <expression (IN) of string>] ',']
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Buttons ':=' <expression (IN) of buttondata>]
  | ['\' BtnArray ':=' <array {*} (IN) of string>]
  ['\' DefaultBtn ':=' <expression (IN) of btnres>]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>] ')'
```

Funkce s vrácenou hodnotou datového typu btnres.

Související informace

Pro informace o	Viz
Data ikony displeje	*icondata - Data ikony displeje na str 1512*
Data tlačítka	*buttondata - Data tlačítka na str 1444*
Výsledková data tlačítka	*btnres - Výsledková data tlačítka na str 1441*
Základní typ boxu zpráv uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Numerický vstup uživatelské interakce	*UINumEntry - Uživatelský číselný vstup na str 1417*
Numerické ladění uživatelské interakce	*UINumTune - Ladění uživatelských čísel na str 1423*
Alfa vstup uživatelské interakce	*UIAlphaEntry - Uživatelský vstup na str 1382*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Systém připojen k FlexPendantu atd.	*UIClientExist - Existence uživatelského klienta na str 1388*
Rozhraní FlexPendantu	*Specifikace produktu - Controller software IRC5*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.195 UINumEntry - Uživatelský číselný vstup

Použití

`UINumEntry` (*User Interaction Number Entry*) se používá pro umožnění operátorovi vložit numerickou hodnotu z dostupného uživatelského zařízení, jako je FlexPendant. Zpráva je napsána pro operátora, který odpoví numerickou hodnotou. Numerická hodnota je potom zkontrolována, schválena a přenesena zpět do programu.

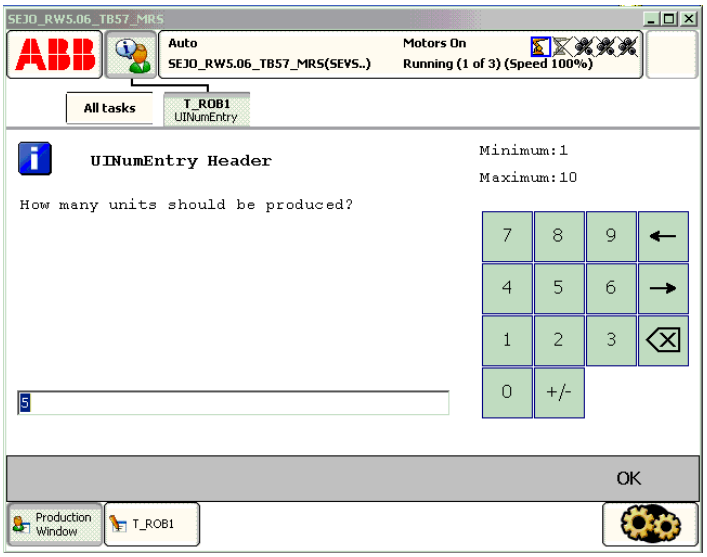
Základní příklady

Následující příklad názorně ukazuje funkci`UINumEntry`.

Viz také *Další příklady na str 1420*.

Příklad 1

```
VAR num answer;
...
answer := UINumEntry(
  \Header:="UINumEntry Header"
  \Message:="How many units should be produced?"
  \Icon:=iconInfo
  \InitValue:=5
  \MinValue:=1
  \MaxValue:=10
  \AsInteger);
FOR i FROM 1 TO answer DO
  produce_part;
ENDFOR
```



xx0500002412

Nahoře, okénko pro numerickou zprávu s ikonou, hlavičkou, zprávou, init-, max- a minvalue zapsanými na displej FlexPendantu. Okénko zpráv kontroluje, aby operátor zvolil celé číslo v rámci rozpětí hodnoty. Vykonávání programu čeká na

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.195 UINumEntry - Uživatelský číselný vstup
RobotWare - OS
Pokračování

stisknutí OK a potom je vrácena zvolená numerická hodnota. Rutina `produce_part` je potom opakována podle počtu vstupů přes FlexPendant.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Tato funkce vrací vstupní numerickou hodnotu.

Jestliže se funkce přeruší přes `\BreakFlag`:

- Jestliže je určen parametr `\InitValue`, tato hodnota je vrácena
- Jestliže není určen parametr `\InitValue` , je vrácena hodnota 0.

Jestliže se funkce přeruší přes chybový handler `ERROR`, nebude vrácena žádná hodnota.

Argumenty

```
UINumEntry ( [\Header] [\Message] | [\MsgArray]
     [\Wrap][\Icon][\InitValue] [\MinValue] [\MaxValue]
     [\AsInteger][\MaxTime] [\DIBreak] [\DIPassive] [\DOBreak]
     [\DOPassive] \BreakFlag])
```

`[\Header]`

Datový typ: `string`

Text hlavičky, který bude zapsán v horní části okénka zprávy. Max. 40 znaků.

`[\Message]`

Datový typ: `string`

Jedna textová řádka bude zapsána na displej. Max. 40 znaků.

`[\MsgArray]`

Message Array

Datový typ: `string`

Several text lines from an array to be written on the display.

Pouze jeden z parametrů `\Message`, nebo `\MsgArray` se může použít ve stejnou dobu.

Max. prostor je 9 řádek se 40 znaky na řádku.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 1420*.

Standardně žádná ikona.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

[\InitValue]

Datový typ: num

Počáteční hodnota, která je zobrazena ve vstupním okénku.

[\MinValue]

Datový typ: num

Minimální hodnota pro vracenou hodnotu.

[\MaxValue]

Datový typ: num

Maximální hodnota pro vracenou hodnotu.

[\AsInteger]

Datový typ: switch

Eliminuje desetinnou tečku z numerické klávesnice, aby bylo zajištěno, že vracená hodnota bude celé číslo.

[\MaxTime]

Datový typ: num

Max množství času v sekundách, kdy vykonávání programu čeká. Jestliže během této doby není stisknuta klávesa OK, program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_MAXTIME může být použita pro testování, jestli uplynula max doba nebo nikoliv.

[\DIBreak]

Digital Input Break

Datový typ: signaldi

Digitální vstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto tlačítko OK před nastavením signálu na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit BreakFlag (viz dole). Konstanta ERR_TP_DIBREAK může být použita pro testování, jestli toto vzniklo nebo nikoliv.

[\DIPassive]

Digital Input Passive

Datový typ: switch

Tento přepínač potlačuje standardní chování při použití volitelného argumentu DIBreak. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný BreakFlag), když je signál DIBreak nastaven na 0 (nebo již je 0). Konstantu ERR_TP_DIBREAK je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

[\DOBreak]

Digital Output Break

Datový typ: signaldo

Digitální výstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto tlačítko OK před nastavením signálu na 1 (nebo již je 1), program

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DOBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DOPassive]`

Digital Output Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DOBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DOBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DOBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\BreakFlag]`

Datový typ: `errnum`

Proměnná (před použitím je nastavena systémem na 0), která bude držet chybový kód, jestliže je použit`\MaxTime`, `\DIBreak`, nebo`\DOBreak`. Konstanty `ERR_TP_MAXTIME`, `ERR_TP_DIBREAK` a `ERR_TP_DOBREAK` mohou být použity pro volbu příčiny. Jestliže tato volitelná proměnná není použita, potom bude vykonán chybový handler.

Vykonávání programu

Numerické okénko pro zprávu s numerickou klávesnicí, ikonou, hlavičkou, řádkami pro zprávu a init-, max- a minvalue se zobrazí podle naprogramovaných argumentů. Vykonávání programu čeká, až uživatel vloží schválenou numerickou hodnotu a stiskne OK nebo až je okénko zpráv přerušeno vypršením času nebo činností signálu. Vstupní numerická hodnota a důvod přerušení jsou přeneseny zpět do programu.

Nové okénko pro zprávu na úrovni TRAP přebírá prioritu od okénka pro zprávu na základní úrovni.

Předdefinovaná data

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
```

Další příklady

Následující příklad názorně ukazuje funkci`UINumEntry`.

Příklad 1

```
VAR errnum err_var;
VAR num answer;
VAR num distance;
...
answer := UINumEntry (\Header:= "BWD move on path"
  \Message:="Enter the path overlap ?" \Icon:=iconInfo
  \InitValue:=5 \MinValue:=0 \MaxValue:=10
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  \MaxTime:=60 \DIBreak:=di5 \BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer distance := 5;
  CASE 0
    ! Operator answer
    distance := answer;
  DEFAULT:
    ! Not such case defined
ENDTEST
```

Okénko pro zprávu je zobrazeno a operátor může vložit numerickou hodnotu a stisknout OK. Okénko zpráv může být také přerušeno s vypršením času nebo digitálním vstupním signálem. V programu je možné vyhledat důvod a vykonat příslušnou činnost.

Řešení chyb

Jestliže parametr `\BreakFlag` není použit, tyto situace mohou být potom ošetřeny chybovým handlerem:

- Jestliže se objevilo vypršení času (parametr `\MaxTime`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_MAXTIME` a vykonávání pokračuje v chybovém handleru.
- Jestliže je nastaven digitální vstup (parametr `\DIBreak`) před vstupem operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DIBREAK` a vykonávání pokračuje v chybovém handleru.
- Jestliže digitální výstup nastaven (parametr `\DOBreak`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DOBREAK` a vykonávání pokračuje v chybovém handleru.

Tato situace může být potom ošetřena chybovým handlerem:

- Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`, `ERRNO` je nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.
- Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná `ERRNO` je nastavena na `ERR_TP_NO_CLIENT` a vykonávání pokračuje v chybovém handleru.
- Počáteční hodnota (parametr `\InitValue`) není určena v rámci rozsahu min a max hodnoty (parametry `\MinValue` a `\MaxValue`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UI_INITVALUE` a vykonávání pokračuje v chybovém handleru.
- Jestliže minimální hodnota (parametr `\MinValue`) je větší než max hodnota (parametr `\MaxValue`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UI_MAXMIN` a vykonávání pokračuje v chybovém handleru.
- Jestliže počáteční hodnota (parametr `\InitValue`) není celé číslo, jak je určeno v parametru `\AsInteger`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UI_NOTINT` a vykonávání pokračuje v chybovém handleru.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.195 UINumEntry - Uživatelský číselný vstup

RobotWare - OS

Pokračování

Omezení

Vyhněte se používání příliš malých hodnot pro parametr vypršení času `\MaxTime`, když je často vykonáván `UINumEntry`, např. ve smyčce. Výsledkem může být nepředvídané chování činnosti systému, jako je zpomalení odezvy FlexPendantu.

Syntaxe

```
UINumEntry '('
  ['\' Header ':=' <expression (IN) of string>]
  [Message ':=' <expression (IN) of string> ]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' InitValue ':=' <expression (IN) of dnum>]
  ['\' MinValue ':=' <expression (IN) of dnum>]
  ['\' MaxValue ':=' <expression (IN) of dnum>]
  ['\' AsInteger]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>] ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Data ikony displeje	*icondata - Data ikony displeje na str 1512*
Základní typ boxu zpráv uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Pokročilý typ boxu zpráv uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*
Numerické ladění uživatelské interakce	*UINumTune - Ladění uživatelských čísel na str 1423*
Alfa vstup uživatelské interakce	*UIAlphaEntry - Uživatelský vstup na str 1382*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Systém připojen k FlexPendantu atd.	*UIClientExist - Existence uživatelského klienta na str 1388*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.196 UINumTune - Ladění uživatelských čísel

Použití

`UINumTune` (*User Interaction Number Tune*) se používá pro umožnění operátorovi ladit numerickou hodnotu z dostupného uživatelského zařízení, jako je FlexPendant. Zpráva je napsána pro operátora, který ladí numerickou hodnotu. Naladěná numerická hodnota je potom zkontrolována, schválena a přenesena zpět do programu.

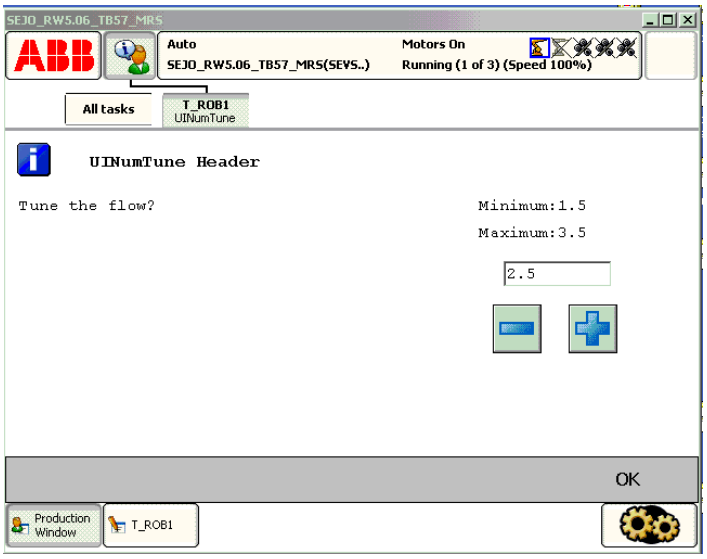
Základní příklady

Následující příklad názorně ukazuje funkci`UINumTune`.

Viz také *Další příklady na str 1426*.

Příklad 1

```
VAR num flow;
...
flow := UINumTune(
  \Header:="UINumTune Header"
  \Message:="Tune the flow?"
  \Icon:=iconInfo,
  2.5,
  0.1
  \MinValue:=1.5
  \MaxValue:=3.5);
```



xx0500002414

Nahoře, okénko pro zprávy numerického ladění s ikonou, hlavičkou, zprávou, init-, přírůstkovou, max- a minvalue zapsanými na displej FlexPendantu. Okénko zpráv kontroluje, že operátor ladí hodnotu flow s krokem 0,1 od init hodnoty 2,5 a drží se v rozsahu hodnoty 1,5 .. 3,5. Vykonávání programu čeká na stisknutí OK a potom je vrácena zvolená numerická hodnota a uložena do proměnné flow.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.196 UINumTune - Ladění uživatelských čísel

RobotWare - OS

Pokračování

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Tato funkce vrací naladěnou numerickou hodnotu.

Jestliže se funkce přeruší přes `\BreakFlag`, je vrácena určená `InitValue`.

Jestliže se funkce přeruší přes chybový handler `ERROR`, nebude vrácena žádná hodnota.

Argumenty

```
UINumTune ( [\Header] [\Message] | [\MsgArray] [\Wrap] [\Icon]
     InitValue Increment [\MinValue] [\MaxValue] [\MaxTime]
     [\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive] [\BreakFlag]
     )
```

`[\Header]`

Datový typ: `string`

Text hlavičky, který bude zapsán v horní části okénka zprávy. Max. 40 znaků.

`[\Message]`

Datový typ: `string`

Jedna textová řádka bude zapsána na displej. Max. 40 znaků.

`[\MsgArray]`

Message Array

Datový typ: `string`

Několik textových řádek od pole, které budou zapsány na displej.

Pouze jeden z parametrů `\Message`, nebo `\MsgArray` se může použít ve stejnou dobu.

Max. prostor je 11 řádek se 40 znaky na řádku.

`[\Wrap]`

Datový typ: `switch`

Jestliže je tak zvoleno, všechny určené řetězce v argumentu `\MsgArray` budou sloučeny do jednoho řetězce s jednoduchou mezerou mezi každým jednotlivým řetězcem a budou rozšířeny na nejmenší počet řádek, jak je to možné.

Standardně bude každý řetězec v argumentu `\MsgArray` na samostatné řádce na displeji.

`[\Icon]`

Datový typ: `icondata`

Definuje ikonu, která bude zobrazena. Může se použít pouze jedna z předdefinovaných ikon typu `icondata`. Viz *Předdefinovaná data na str 1426*.

Standardně žádná ikona.

`InitValue`

Datový typ: `num`

Počáteční hodnota, která je zobrazena ve vstupním okénku.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`Increment`

Datový typ: `num`

Tento parametr určuje, jak hodně by se hodnota měla změnit, když je stisknuto tlačítko plus nebo minus.

`[\MinValue]`

Datový typ: `num`

Minimální hodnota pro vracenou hodnotu.

`[\MaxValue]`

Datový typ: `num`

Maximální hodnota pro vracenou hodnotu.

`[\MaxTime]`

Datový typ: `num`

Max množství času v sekundách, kdy vykonávání programu čeká. Jestliže během této doby není stisknuta klávesa OK, program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_MAXTIME` může být použita pro testování, jestli uplynula max doba nebo nikoliv.

`[\DIBreak]`

Digital Input Break

Datový typ: `signaldi`

Digitální vstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto tlačítko OK před nastavením signálu na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DIBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

`[\DIPassive]`

Digital Input Passive

Datový typ: `switch`

Tento přepínač potlačuje standardní chování při použití volitelného argumentu `DIBreak`. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný `BreakFlag`), když je signál `DIBreak` nastaven na 0 (nebo již je 0). Konstantu `ERR_TP_DIBREAK` je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

`[\DOBreak]`

Digital Output Break

Datový typ: `signaldo`

Digitální výstupní signál, který může přerušit dialog operátora. Jestliže není stisknuto tlačítko OK před nastavením signálu na 1 (nebo již je 1), program pokračuje vykonávání v chybovém handleru, pokud není použit `BreakFlag` (viz dole). Konstanta `ERR_TP_DOBREAK` může být použita pro testování, jestli toto vzniklo nebo nikoliv.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.196 UINumTune - Ladění uživatelských čísel

RobotWare - OS

Pokračování

[\DOPassive]

Digital Output Passive

Datový typ: switch

Tento přepínač potlačuje standardní chování při použití volitelného argumentu DOBreak. Namísto reagování, když signál je nastaven na 1 (nebo je již 1) by instrukce měla pokračovat v chybovém handleru (jestliže se nepoužívá žádný BreakFlag), když je signál DOBreak nastaven na 0 (nebo již je 0). Konstantu ERR_TP_DOBREAK je možné použít pro testování, jestli už toto nastalo nebo nikoliv.

[\BreakFlag]

Datový typ: errnum

Proměnná (před použitím je nastavena systémem na 0), která bude držet chybový kód, jestliže je použit\MaxTime, \DIBreak, nebo\DOBreak. Konstanty ERR_TP_MAXTIME, ERR_TP_DIBREAK a ERR_TP_DOBREAK mohou být použity pro volbu příčiny. Jestliže tato volitelná proměnná není použita, potom bude vykonán chybový handler.

Vykonávání programu

Okénko zpráv numerického ladění s ladicími tlačítky +/-, ikonou, hlavičkou, řádkami pro zprávu, init-, přírůstkovou, max- a minvalue se zobrazí podle naprogramovaných argumentů. Vykonávání programu čeká, až uživatel vyladí numerickou hodnotu a stiskne OK nebo až je okénko zpráv přerušeno vypršením času nebo činností signálu. Vstupní numerická hodnota a důvod přerušení jsou přeneseny zpět do programu.

Nové okénko pro zprávu na úrovni TRAP přebírá prioritu od okénka pro zprávu na základní úrovni.

Předdefinovaná data

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
```

Další příklady

Následující příklad názorně ukazuje funkciUINumTune.

Příklad 1

```
VAR errnum err_var;
VAR num tune_answer;
VAR num distance;
...
tune_answer := UINumTune (\Header:=" BWD move on path"
     \Message:="Enter the path overlap ?" \Icon:=iconInfo, 5, 1
     \MinValue:=0 \MaxValue:=10 \MaxTime:=60 \DIBreak:=di5
     \BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
    CASE ERR_TP_DIBREAK:
      ! No operator answer
      distance := 5;
    CASE 0:
      ! Operator answer
      distance := tune_answer;
    DEFAULT:
      ! No such case defined
  ENDTEST
```

Okénko pro zprávu ladění je zobrazeno a operátor může ladit numerickou hodnotu a stisknout OK. Okénko zpráv může být také přerušeno s vypršením času nebo digitálním vstupním signálem. V programu je možné vyhledat důvod a vykonat příslušnou činnost.

Řešení chyb

Jestliže parametr `\BreakFlag` není použit, tyto situace mohou být potom ošetřeny chybovým handlerem:

- Jestliže se objevilo vypršení času (parametr `\MaxTime`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_MAXTIME` a vykonávání pokračuje v chybovém handleru.
- Jestliže je nastaven digitální vstup (parametr `\DIBreak`) před vstupem operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DIBREAK` a vykonávání pokračuje v chybovém handleru.
- Jestliže digitální výstup nastaven (parametr `\DOBreak`) před vstupem od operátora, systémová proměnná `ERRNO` je nastavena na `ERR_TP_DOBREAK` a vykonávání pokračuje v chybovém handleru.

Tato situace může být potom ošetřena chybovým handlerem:

- Jestliže proměnná signálu je proměnná deklarovaná v RAPIDu a nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`, `ERRNO` je nastavena na `ERR_NO_ALIASIO_DEF` a vykonávání pokračuje v chybovém handleru.
- Jestliže neexistuje žádný klient, např. FlexPendant, který by se postaral o instrukci, systémová proměnná `ERRNO` je nastavena na `ERR_TP_NO_CLIENT` a vykonávání pokračuje v chybovém handleru.
- Počáteční hodnota (parametr `\InitValue`) není určena v rámci rozsahu min a max hodnoty (parametry `\MinValue` a `\MaxValue`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UI_INITVALUE` a vykonávání pokračuje v chybovém handleru.
- Jestliže minimální hodnota (parametr `\MinValue`) je větší než max hodnota (parametr `\MaxValue`), potom je systémová proměnná `ERRNO` nastavena na `ERR_UI_MAXMIN` a vykonávání pokračuje v chybovém handleru.

Omezení

Vyhněte se používání příliš malých hodnot pro parametr vypršení času `\MaxTime`, když je často vykonáván `UINumTune`, např. ve smyčce. Výsledkem může být nepředvídané chování činnosti systému, jako je zpomalení odezvy FlexPendantu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.196 UINumTune - Ladění uživatelských čísel

RobotWare - OS

Pokračování

Syntaxe

```
UINumTune '('
  ['\' Header ':=' <expression (IN) of string>]
  [Message ':=' <expression (IN) of string> ]
  | ['\' MsgArray ':='<array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  [InitValue ':=' <expression (IN) of num>]
  [Increment ':=' <expression (IN) of num>]
  ['\' MinValue ':=' <expression (IN) of num>]
  ['\' MaxValue ':=' <expression (IN) of num>]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>] ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Data ikony displeje	*icondata - Data ikony displeje na str 1512*
Základní typ boxu zpráv uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Pokročilý typ boxu zpráv uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*
Numerický vstup uživatelské interakce	*UINumEntry - Uživatelský číselný vstup na str 1417*
Alfa vstup uživatelské interakce	*UIAlphaEntry - Uživatelský vstup na str 1382*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Systém připojen k FlexPendantu atd.	*UIClientExist - Existence uživatelského klienta na str 1388*
Vyčistit okno operátora	*TPErase - Vymaže text vytištěný na FlexPendantu na str 781*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.197 ValidIO - Platný I/O signál k přístupu

Použití

`ValidIO` se používá ke kontrole, jestli je momentálně přístup k určenému I/O signálu bez jakékoliv chyby.

Základní příklady

Následující příklad názorně ukazuje funkci`ValidIO`.

Příklad 1

```
IF ValidIO(mydosignal) SetDO mydosignal, 1;
```

Nastavte digitální výstupní signál `mydosignal` na `1`, jestliže jeho I/O jednotka je zapnuta a běží.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `bool`

Vrací `TRUE`, jestliže I/O signál je platný a I/O jednotka pro signál je zapnuta a běží.

Vrací `FALSE`, jestliže I/O jednotka není zapnuta a neběží, nebo když nebyla vykonána žádná instrukce `AliasIO` pro připojení proměnné signálu deklarované v programu RAPID k signálu definovaném v I/O konfiguraci.

Argumenty

```
ValidIO (Signal)
```

`Signal`

Datový typ: `signalxx`

Jméno I/O signálu. Musí být datového typu `signaldo`, `signaldi`, `signalgo`, `signalgi`, `signalao` nebo `signalai`.

Vykonávání programu

Chování provádění:

- Zkontrolujte, jestli I/O signál je platný
- Zkontrolujte, jestli I/O jednotka pro signál je zapnuta a běží.

Nejsou generovány žádné chybové zprávy.

Syntaxe

```
ValidIO '('
  [Signal ':='] <variable (VAR) of anytype> ')'
```

Funkce s vrácenou hodnotou datového typu `bool`.

Související informace

Pro informace o	Viz
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.197 ValidIO - Platný I/O signál k přístupu

RobotWare - OS

Pokračování

Pro informace o	Viz
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*
Definovat I/O signál se jménem aliasu	*AliasIO - Definovat V/V (I/O) signál se jménem aliasu na str 28*
Přečíst atribut systémového parametru	*ReadCfgData - Čte atribut systémového parametru na str 523*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.198 ValToStr - Převádí hodnotu na řetězec

Použití

`ValToStr` (*Value To String*) se používá k převodu hodnoty jakéhokoliv datového typu na řetězec.

Základní příklady

Následující příklady názorně ukazují funkci`ValToStr`.

Příklad 1

```
VAR string str;
VAR pos p := [100,200,300];
str := ValToStr(p);
```

Proměnné `str` je dána hodnota `"[100,200,300]"`.

Příklad 2

```
str := ValToStr(TRUE);
```

Proměnné `str` je dána hodnota `TRUE`.

Příklad 3

```
str := ValToStr(1.234567890123456789);
```

Proměnné `str` je dána hodnota `"1.23456789012346"`.

Příklad 4

```
VAR num numtype:=1.234567890123456789;

str := ValToStr(numtype);
```

Proměnné `str` je dána hodnota `"1.23457"`.

Příklad 5

```
VAR dnum dnumtype:=1.234567890123456789;

str := ValToStr(dnumtype);
```

Proměnné `str` je dána hodnota `"1.23456789012346"`.

Vratná hodnota (Vrátit hodnotu)

Datový typ: `string`

Hodnota je převedena na řetězec se standardním formátem `RAPID`. To znamená v principu 6 podstatných číslic. Literální hodnota interpretovaná jako proměnné `dnum` (viz příklad 3) a `dnum` (viz příklad 5) má přesto 15 podstatných číslic.

Chyba za běhu je generována, jestliže výsledný řetězec je příliš dlouhý.

Argumenty

```
ValToStr ( Val )
```

`Val`

Value

Datový typ: `anytype`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.198 ValToStr - Převádí hodnotu na řetězec

RobotWare - OS

Pokračování

Hodnota všech datových typů. Všechny typy hodnotových dat se strukturou atomická, záznam, komponent záznamu, pole nebo prvek pole se mohou používat.

Syntaxe

```
ValToStr '('
  [ Val ':=' ] <expression (IN) of anytype> ')'
```

Funkce s vrácenou hodnotou datového typu `string`.

Související informace

Pro informace o	Viz
Funkce s řetězci	*Technická referenční příručka - Přehled RAPID*
Definice řetězce	*string (řetězec) - Řetězce na str 1596*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.199 VectMagn - Magnituda pos vektoru

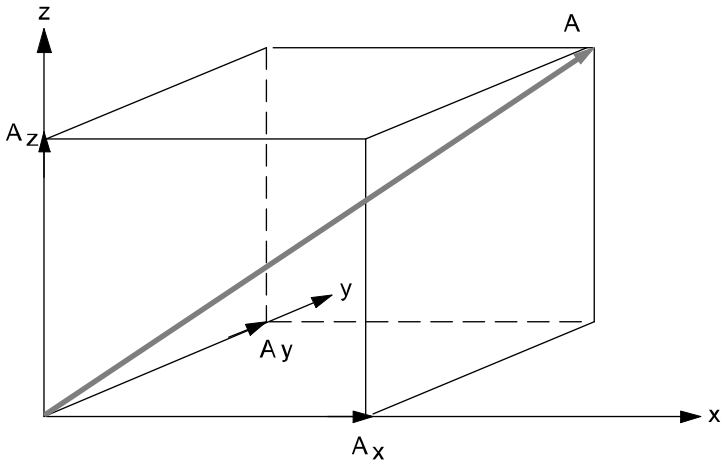
Použití

`VectMagn` (***Vector Magnitude***) se používá pro výpočet magnitudy `pos` vektoru.

Základní příklady

Následující příklad názorně ukazuje funkci`VectMagn`.

Příklad 1



xx0500002446

Vektor **A** může být zapsán jako součet jeho komponentů ve třech ortogonálních směrech:

$A=A_x x+A_y y+A_y y$

Magnituda **A** je:

$$|A| = \sqrt{A_x^2 + A_y^2 + A_z^2}$$

Vektor je popsán datovým typem `pos` a magnituda datovým typem `num`:

```
VAR num magnitude;
VAR pos vector;
...
vector := [1,1,1];
magnitude := VectMagn(vector);
```

Vratná hodnota (Vrátit hodnotu)

Datový typ: `num`

Magnituda vektoru (datový typ `pos`).

Argumenty

```
VectMagn (Vector)
```

`Vector`

Datový typ: `pos`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.199 VectMagn - Magnituda pos vektoru

RobotWare - OS

Pokračování

Vektor popsaný datovým typem `pos`.

Syntaxe

```
VectMagn '('
  [Vector ':='] <expression (IN) of pos> ')'
```

Funkce s vrácenou hodnotou datového typu `num`.

Související informace

Pro informace o	Viz
Matematické instrukce a funkce	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

2.200 XOR - Vyhodnocuje logickou hodnotu

Použití

XOR (*Exclusive Or*) je podmíněný výraz, který se používá k hodnocení logické hodnoty (true/false).

Základní příklady

Následující příklady názorně ukazují funkci XOR.

Příklad 1

```
VAR bool a;
VAR bool b;
VAR bool c;
c := a XOR b;
```

Vrácená hodnota c je TRUE, jestliže jedna, a pouze jedna z a nebo b jsou TRUE. Jinak je vrácena hodnota FALSE.

Příklad 2

```
VAR num a;
VAR num b;
VAR bool c;
...
c := a>5 XOR b=3;
```

Vrácená hodnota c je TRUE, jestliže jedna, a pouze jedna z podmínek je TRUE. Buď a je větší než 5 nebo b je rovno 3. Jinak je vrácená hodnota FALSE.

Vratná hodnota (Vrátit hodnotu)

Datový typ: bool

Vrácená hodnota je TRUE, jestliže jeden, a pouze jeden z podmíněných výrazů je správný. Jinak je vrácená hodnota FALSE.

Syntaxe

```
<expression of bool> XOR <expression of bool>
```

Funkce s vrácenou hodnotou datového typu bool.

Související informace

Pro informace o	Viz
AND	*AND - Vyhodnocuje logickou hodnotu na str 1035*
OR	*OR - Vyhodnocuje logickou hodnotu na str 1242*
NOT	*NOT - Invertuje logickou hodnotu na str 1233*
Výrazy	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Tato stránka je záměrně prázdná

3 Datové typy

3.1 aiotrigg - Analogová I/O trigger podmínka

Použití

`aiotrigg` (*Analog I/O Trigger*) se používá k definování podmínky pro generování přerušení pro analogový vstupní nebo výstupní signál.

Popis

Data typu `aiotrigg` definují způsob, jak bude použit nízký a vysoký práh pro určení, jestli logická hodnota analogového signálu splňuje podmínku pro generování přerušení.

Základní příklady

Následující příklad názorně ukazuje datový typ `aiotrigg`:

Příklad 1

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalAI \Single, ai1, AIO_BETWEEN, 1.5, 0.5, 0, sig1int;
```

Přikazuje přerušení, které se objeví poprvé, když logická hodnota analogového vstupního signálu `ai1` je mezi `0.5` a `1.5`. Potom je provedeno volání trap rutiny `iroutine1` .

Předdefinovaná data

Následující symbolické konstanty datového typu `aiotrigg` jsou předdefinovány a mohou se používat k určení podmínky pro instrukce `ISignalAI` a `ISignalAO`.

Hodnota	Symbolická konstanta	Komentář
1	`AIO_ABOVE_HIGH`	Signál bude generovat přerušení, jestliže je nad určenou vysokou hodnotou
2	`AIO_BELOW_HIGH`	Signál bude generovat přerušení, jestliže je pod určenou hodnotou
3	`AIO_ABOVE_LOW`	Signál bude generovat přerušení, jestliže je nad určenou nízkou hodnotou
4	`AIO_BELOW_LOW`	Signál bude generovat přerušení, jestliže je pod určenou nízkou hodnotou
5	`AIO_BETWEEN`	Signál bude generovat přerušení, jestliže je mezi určenou nízkou a vysokou hodnotou
6	`AIO_OUTSIDE`	Signál bude generovat přerušení, jestliže je pod určenou nízkou hodnotou a nad určenou vysokou hodnotou
7	`AIO_ALWAYS`	Signál bude vždy generovat přerušení

Charakteristika

`aiotrigg` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.1 aiotrigg - Analogová I/O trigger podmínka

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Přerušení od analogového vstupního signálu	*ISignalAI - Přerušuje od analogového vstupního signálu na str 292*
Přerušení od analogového výstupního signálu	*ISignalAO - Přerušuje od analogového výstupního signálu na str 302*
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.2 ALIAS - Přidělování datového typu alias

Použití

ALIAS se používá k definování datového typu rovnocenného s jiným datovým typem. Typy alias poskytují prostředky pro klasifikaci objektů. Systém může používat klasifikaci alias pro vyhledání a uvedení objektů se vztahem k typu. Typ alias je uveden definicí alias.

Vestavěné typy alias jsou errnum a intnum, oba alternují za num.

typ errnum

Typ errnum je alias pro num a používá se pro reprezentaci chybových čísel.

typ intnum

Typ intnum je alias pro num a používá se pro reprezentaci čísel přerušení.

Základní příklady

Následující příklad názorně ukazuje definici ALIAS.

Příklad 1

```
ALIAS num level;
CONST level low := 2.5;
CONST level high := 4.0;
```

Aliasový typ level je definován (alias pro num).

Omezení

Aby mohly být rozpoznány RAPIDem, všechny definice alias musí být deklarovány na úplném vrcholu programu nebo systémového modulu před deklaracemi všech ostatních dat. Jediný datový typ, kterému je povoleno být deklarován přes aliasem, je RECORD.

Jeden aliasový typ nemůže být definován nad jiným aliasovým typem.

Syntaxe

```
ALIAS <type name> <identifier> ';'
```

Definice aliasu.

Související informace

Pro informace o	Viz
errnum - Chybové číslo	*errnum - Chybové číslo na str 1495*
intnum - Identita přerušení	*intnum - Identita přerušení na str 1516*
slovní prvky	*Technická referenční příručka - RAPID kernel*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.3 bool - Logické hodnoty

Použití

`bool` se používá pro logické hodnoty (true/false).

Popis

Hodnota dat typu `bool` může být buď `TRUE` nebo `FALSE`.

Základní příklady

Následující příklady názorně ukazují datový typ `bool` :

Příklad 1

```
flag1 := TRUE;
```

příznaku je přidělena hodnota TRUE.

Příklad 2

```
VAR bool highvalue;
VAR num reg1;
...
highvalue := reg1 > 100;
```

Pro `highvalue` je přidělena hodnota `TRUE`, jestliže `reg1` je větší než `100`; jinak je přidělena `FALSE`.

Příklad 3

```
IF highvalue Set do1;
```

Signál `do1` se nastavuje, když `highvalue` je `TRUE`.

Příklad 4

```
highvalue := reg1 > 100;
mediumvalue := reg1 > 20 AND NOT highvalue;
```

Pro `mediumvalue` je přidělena hodnota `TRUE`, jestliže `reg1` je mezi `20` a `100`.

Související informace

Pro informace o	Viz
Logické výrazy	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Výrazy*
Operace pomocí logických hodnot	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Výrazy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.4 btnres - Výsledková data tlačítka

Použití

`btnres` (*button result*) se používá pro reprezentování výběru uživatele na tlačítkovém displeji uživatelského zařízení, jako je FlexPendant.

Popis

Konstanta `btnres` je určena pro použití při kontrole výsledkové hodnoty od instrukce `UIMsgBox` a vrácené hodnoty od funkcí `UIMessageBox` a `UIListView`.

Základní příklady

Následující příklad názorně ukazuje datový typ `btnres`:

Příklad 1

```
VAR btnres answer;

UIMsgBox "More ?" \Buttons:=btnYesNo \Result:= answer;
IF answer= resYes THEN
  ...
ELSEIF answer =ResNo THEN
  ...
ENDIF
```

Standardní výčet tlačítek `btnYesNo` bude dávat jedno tlačítko Yes a jedno tlačítko No na uživatelském rozhraní. Výběr uživatele bude uložen do proměnné `answer`.

Předdefinovaná data

Následující konstanty datového typu `btnres` jsou předdefinovány v systému

Hodnota	Konstanty	Odpověď tlačítka
0	`resUnkwn`	Neznámý výsledek
1	`resOK`	OK
2	`resAbort`	Preruit
3	`resRetry`	Zkuste to znovu
4	`resIgnore`	Ignor.
5	`resCancel`	Storno
6	`resYes`	Ano
7	`resNo`	No

Je možné pracovat s uživatelsky definovanými tlačítky, která odpovídají s funkcemi `UIMessageBox` a `UIListView`.

Charakteristika

`btnres` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.4 btnres - Výsledková data tlačítka

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Box pro zprávy uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Box pro zprávy uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Data tlačítka aliasového datového typu	*buttondata - Data tlačítka na str 1444*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.5 busstate - Stav I/O sběrnice

Použití

`busstate` se používá pro zrcadlení stavu, ve kterém se I/O sběrnice aktuálně nachází.

Popis

Konstanta `busstate` je určena k použití při kontrole vracené hodnoty od instrukce `IOBusState`.

Základní příklady

Následující příklad názorně ukazuje datový typ `busstate`:

Příklad 1

```
VAR busstate bstate;

IOBusState "IBS", bstate \Phys;
TEST bstate
CASE IOBUS_PHYS_STATE_RUNNING:
  ! Possible to access some signal on the IBS bus
DEFAULT:
  ! Actions for not up and running IBS bus
ENDTEST
```

Předdefinovaná data

Předdefinované symbolické konstanty datového typu `busstate` je možné prohlížet v instrukci `IOBusState`.

Charakteristika

`busstate` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Získat aktuální stav I/O sběrnice	*IOBusState - Získat aktuální stav I/O sběrnice na str 275*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.6 buttondata - Data tlačítka

Použití

`buttondata` se používá pro reprezentování standardní kombinace tlačítek pro displej uživatelského zařízení, jako je FlexPendant.

Popis

Konstanta `buttondata` se používá pro reprezentování tlačítek odezvy v instrukci `UIMsgBox` a funkcích `UIMessageBox` a `UIListView`.

Základní příklady

Následující příklad názorně ukazuje datový typ `buttondata`:

Příklad 1

```
VAR btnres answer;
UIMsgBox "More ?" \Buttons:=btnYesNo \Result:= answer;
IF answer= resYes THEN
...
ELSE
...
ENDIF
```

Standardní výčet tlačítek `btnYesNo` bude dávat jedno tlačítko Yes a jedno tlačítko No.

Předdefinovaná data

Následující konstanty datového typu `buttondata` jsou předdefinovány v systému.

Hodnota	Konstanty	Zobrazené tlačítko
- 1	`btnNone`	Žádné tlačítko
0	`btnOK`	OK
1	`btnAbrtRtryIgn`	Přerušit, zkusit znovu a ignorovat
2	`btnOKCancel`	OK a Zrušit (Cancel)
3	`btnRetryCancel`	Zkusit znovu a Zrušit
4	`btnYesNo`	Ano a Ne (Yes a No)
5	`btnYesNoCancel`	Ano, Ne a Zrušit (Yes, No a Cancel)

Je možné zobrazovat uživatelsky definovaná tlačítka s funkcemi `UIMessageBox` a `UIListView`.

Charakteristika

`buttondata` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Box pro zprávy uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Box pro zprávy uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Výsledek tlačítka aliasového datového typu	*btnres - Výsledková data tlačítka na str 1441*
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.7 bajt - Hodnoty celého čísla 0 - 255

Použití

`byte` se používá pro hodnoty celého čísla (0 - 255) podle rozsahu bajtu.

Tento datový typ se používá ve spojení s instrukcemi a funkcemi, které zpracovávají bitové manipulace a převádějí funkce.

Popis

Data typu `byte` reprezentují bajtovou hodnotu celého čísla.

Základní příklady

Následující příklady názorně ukazují datový typ `byte`:

Příklad 1

```
VAR byte data1 := 130;
```

Definice proměnné `data1` s desítkovou hodnotou 130.

Příklad 2

```
CONST num parity_bit := 8;
VAR byte data1 := 130;
BitClear data1, parity_bit;
```

Bit číslo 8 (`parity_bit`) v proměnné `data1` bude nastaven na 0, např. obsah proměnné `data1` bude změněn ze 130 na 2 (zastoupení celého čísla).

Řešení chyb

Jestliže argument typu `byte` má hodnotu, která není v rozsahu mezi 0 a 255, bude vrácena chyba při vykonávání programu.

Charakteristika

`byte` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*
Bitové funkce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Bitové funkce*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.8 cameradev - kamerové zařízení

Použití

`cameradev` *(camera device)* se používá k definování různých kamerových zařízení s řízením a přístupem z programu RAPID. Datový typ `cameradev` se používá pro instrukce a funkce komunikující s kamerou. Jména kamerových zařízení jsou definována v systémových parametrech a proto nemusí být definována v programu.

Popis

Data typu `cameradev` obsahují pouze reference ke kamerovému zařízení.

Omezení

Data typu `cameradev` nemusí být definována v programu. Nicméně, pokud jsou, bude zobrazena chybová zpráva, jakmile je vykonávána instrukce nebo funkce, která odkazuje k tomuto `cameradev`. Datový typ může, na druhé straně, být použit jako parametr při deklarování rutiny.

Předdefinovaná data

Všechny kamery definované v systémových parametrech jsou předdefinovány v každé programové rutině.

Základní příklady

Následující příklad názorně ukazuje datový typ `cameradev`.

Příklad 1

```
CamLoadJob mycamera, "myjob.job";
```

Charakteristika

`cameradev` je nehodnotový datový typ. To znamená, že data tohoto typu nedovolují operace orientované na hodnotu.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.9 cameratarget - data kamery

Použití

`cameratarget` se používá k výměně vizuálních dat z kamery do programu RAPID.

Popis

Data typu `cameratarget` jsou uživatelsky definovaný soubor dat, který může být nastaven pro výměnu vizuálních dat z kamerového obrazu do programu RAPID.

Data mají rozpětí komponentů, které může být nastaveno podle konkrétních potřeb v aktuální vizuální aplikaci. Komponent `cframe` je zamýšlen pro přenosovou operaci o umístění objektu, zatímco numerické hodnoty a řetězce jsem zamýšleny pro držení inspekčních dat.

Komponenty

Datový typ má následující komponenty:

`name`

Datový typ: `string`

Identifikátor jména `cameratarget`.

`cframe`

current frame

Datový typ: `pose`

Pro ukládání pozičních dat, což se normálně používá pro vedení robotu pomocí modifikace pracovního objektu.

`val1`

value 1

Datový typ: `num`

Pro ukládání numerických výstupů, jako jsou měření.

`...`

`val5`

value 5

Datový typ: `num`

Pro ukládání numerických výstupů, jako jsou měření.

`string1`

Datový typ: `string`

Pro ukládání numerického vizuálního výstupu, jako je výstup inspekce nebo identifikace.

`string2`

Datový typ: `string`

Pro ukládání numerického vizuálního výstupu, jako je výstup inspekce nebo identifikace.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`type`

Datový typ: `num`

Numerický identifikátor kamerového cíle. Podobný účel jako komponent `name`.

`cameraname`

Datový typ: `string`

Jméno kamery

`sceneid`

scene identification

Datový typ: `num`

Jedinečný identifikátor obrazu použitý pro generování `cameratarget`.

Základní příklady

Následující příklad názorně ukazuje datový typ `cameratarget`.

Příklad 1

```
VAR cameratarget target1;
...
wobjmycamera.oframe := target1.cframe;
MoveL pickpart, v100, fine, mygripper \WObj:= wobjmycamera;
```

Souřadnicová transformace `cframe` **je přidělena rámci objektu pracovního objektu.** `robtarget` `pickpart` **byl dříve vyladěn na správnou nabírací pozici v rámci objektu pracovního objektu.**

Konstrukce

```
< dataobject of cameratarget >
  < name of string >
  < cframe of pose >
    < trans of pos >
    < rot of orient >
  < val1 of num >
  < val2 of num >
  < val3 of num >
  < val4 of num >
  < val5 of num >
  < string1 of string >
  < string2 of string >
  < type of num >
  < cameraname of string >
  < sceneid of num >
```


© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.10 capdata - CAP data

Použití

`capdata` obsahuje veškerá data nezbytná pro definování chování CAP procesu.

Komponenty

start_fly

Letmý start

Datový typ: `bool`

Definuje, jestli se používá letmý start nebo nikoliv:

Hodnota	Důsledek
TRUE	letmý start se používá
FALSE	letmý start se NEPOUŽÍVÁ

Letmý start znamená, že pohyb robotu je spuštěn před spuštěním procesu. Proces je potom spuštěn za běhu (viz *flypointdata - Data pro letmý start/konec na str 1508*).

end_fly

Letmý konec

Datový typ: `bool`

Definuje, jestli se používá letmý konec nebo nikoliv:

Hodnota	Důsledek
TRUE	letmý konec se používá
FALSE	letmý konec se NEPOUŽÍVÁ

Letmý konec znamená, že CAP proces může být ukončen předtím, než robot dosáhne koncového bodu, a tím umožňuje robotu opustit procesní dráhu za běhu, tzn. pomocí zónového bodu (viz *flypointdata - Data pro letmý start/konec na str 1508*).

first_instr

První instrukce

Datový typ: `bool`

Definuje, jestli instrukce `CapL`/`CapC` je první instrukcí v sekvenci instrukcí `CapL`/`CapC` nebo nikoliv:

Hodnota	Důsledek
TRUE	toto je první instrukce v sekvenci instrukcí `CapL`/`CapC`
FALSE	toto není první instrukce v sekvenci instrukcí `CapL`/`CapC`

last_instr

Poslední instrukce

Datový typ: `bool`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Definuje, jestli instrukce `CapL/CapC` je poslední instrukcí v sekvenci instrukcí `CapL/CapC` nebo nikoliv:

Hodnota	Důsledek
TRUE	toto je poslední instrukce v sekvenci instrukcí `CapL/CapC`
FALSE	toto není poslední instrukce v sekvenci instrukcí `CapL/CapC`

restart_dist

Vzdálenost restartu, jednotka: mm

Datový typ: `num`

Definuje vzdálenost, kterou se robot musí vrátit podél dráhy, když je restartován po setkání se stopem, když byl CAP proces aktivní.

V systémech MultiMove musí všechny synchronizované roboty používat stejnou restartovací vzdálenost.

speed_data

Rychlostní data pro CAP

Datový typ: `capspeeddata`

Definuje všechna CAP data týkající se rychlosti (viz *capspeeddata - Rychlostní data pro CAP na str 1458*).

start_fly_point

Datový typ: `flypointdata`

Tato data se berou v úvahu pouze když `start_fly` je `TRUE`.

Definuje informace letmého startu pro CAP proces (viz *flypointdata - Data pro letmý start/konec na str 1508*.)

end_fly_point

Datový typ: `flypointdata`

Tato data se berou v úvahu pouze když `start_fly` je `TRUE`.

Definuje informace letmého konce pro CAP proces (viz *flypointdata - Data pro letmý start/konec na str 1508*.)

sup_timeouts

Datový typ: `supervtimeouts`

Definuje vypršení času použitá pro všechny dohledy spojování (viz *supervtimeouts - Časové limity dohledu navázání spojení na str 1599* a sekce *Dohled* v *Application manual - Continuous Application Platform*).

proc_times

Datový typ: `processtimes`

Definuje vypršení času použitá pro fáze supervize statutu PRE, POST1 a POST2 (viz *processtimes - procesní časy na str 1556* a sekce *Dohled a procesní fáze* v *Application manual - Continuous Application Platform*).

block_at_restart

Datový typ: `restartblkdata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.10 capdata - CAP data

Continuous Application Platform (CAP)

Pokračování

Definuje chování CAP procesu během restartu (viz *restartblkdata - bloková data pro restart na str 1561*).

Konstrukce

```
< data object of capdata >
  < start_fly of bool >
  < end_fly of bool >
  < first_instr of bool >
  < last_instr of bool >
  < restart_dist of num >
  < speed_data of capspeeddata >
    < fly_start of num >
    < start of num >
    < startspeed_time of num >
    < startmove_delay of num >
    < main of num >
    < fly_end of num >
  < start_fly_point of flypointdata >
    < from_start of bool >
    < time_before of num >
    < distance of num >
  < end_fly_point of flypointdata >
    < from_start of bool >
    < time_before of num >
    < distance of num >
  < sup_timeouts of supervtimeouts >
    < pre_cond of num >
    < start_cond of num >
    < end_main_cond of num >
    < end_post1_cond of num >
    < end_post2_cond of num >
  < proc_times of processtimes >
    < pre of num >
    < post1 of num >
    < post2 of num >
  < block_at_restart of restartblkdata >
    < weave_start of bool >
    < motion_delay of bool >
    < pre_phase of bool >
    < startspeed_phase of bool >
    < post1_phase of bool >
    < post2_phase of bool >
```

Související informace

	Popsáno v:
Continuous Application Platform	*Application manual - Continuous Application Platform*
Datový typ `capspeeddata`	*capspeeddata - Rychlostní data pro CAP na str 1458*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

	Popsáno v:
Datový typ `flypointdata`	*flypointdata - Data pro letmý start/konec na str 1508*
Datový typ `supervtimeouts`	*supervtimeouts - Časové limity dohledu navázání spojení na str 1599*
Datový typ `processtimes`	*processtimes - procesní časy na str 1556*
Datový typ `block_at_restart`	*restartblkdata - bloková data pro restart na str 1561*
Instrukce `CapL`	*CapL - Instrukce pro lineární pohyb CAP na str 82*
Instrukce `CapC`	*CapC - Pohybová instrukce cirkulárního CAP na str 69*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.11 caplatrackdata - traťová data CAP Look-Ahead-Tracker

Použití

`caplatrackdata` obsahuje data, se kterými může uživatel ovlivňovat, jak instrukce `CapL/CapC` začleňují korekční data dráhy, která generoval Look-Ahead-Tracker (například laserový sledovač). `caplatrackdata` je součástí `captrackdata`.

Komponenty

joint_no

Datový typ: `num`

Definuje typ spoje, vyjádřeného jako číslo, který by mělo používat zařízení senzoru během sledování.

filter

Datový typ: `num`

Definuje časovou konstantu dolní propusti aplikované na korekce dráhy. Komponent může být nastaven na hodnoty mezi 1 a 10, kde 1 dává nejrychlejší odezvu (žádná filtrace) na chyby dráhy zjištěné senzorem.

calibframe_no

Datový typ: `num`

Definuje, který kalibrační rámec ze tří rámců definovaných v `CapLATrSetup` by měl být použit.

Hodnota	Kalibrační rámec	Popis
1	calibframe	Povinné v `CapLATrSetup`
2	calibframe2	Volitelné v `CapLATrSetup`
3	calibframe3	Volitelné v `CapLATrSetup`)

seamoffs_y, seamoffs_z

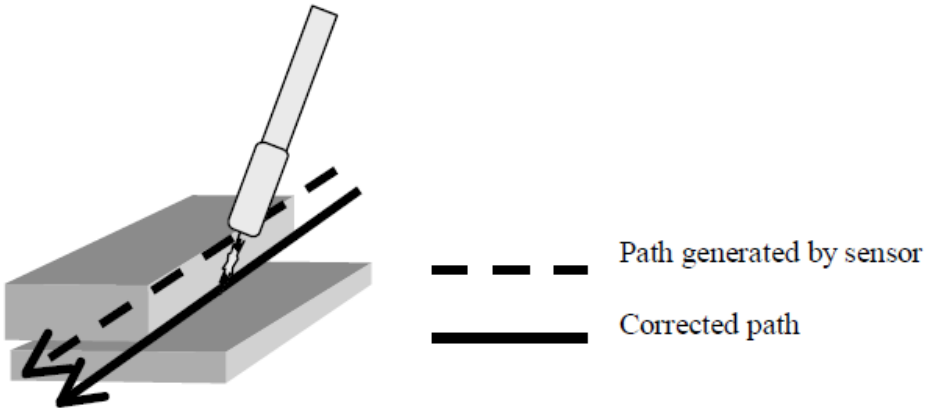
Datový typ: `num`

Komponenty ofsetu švu se používají pro přidání konstantních ofsetů ke dráze generované senzorem (v mm). Jestliže, například, senzor rozhodne, že horní okraj

Pokračování na další straně

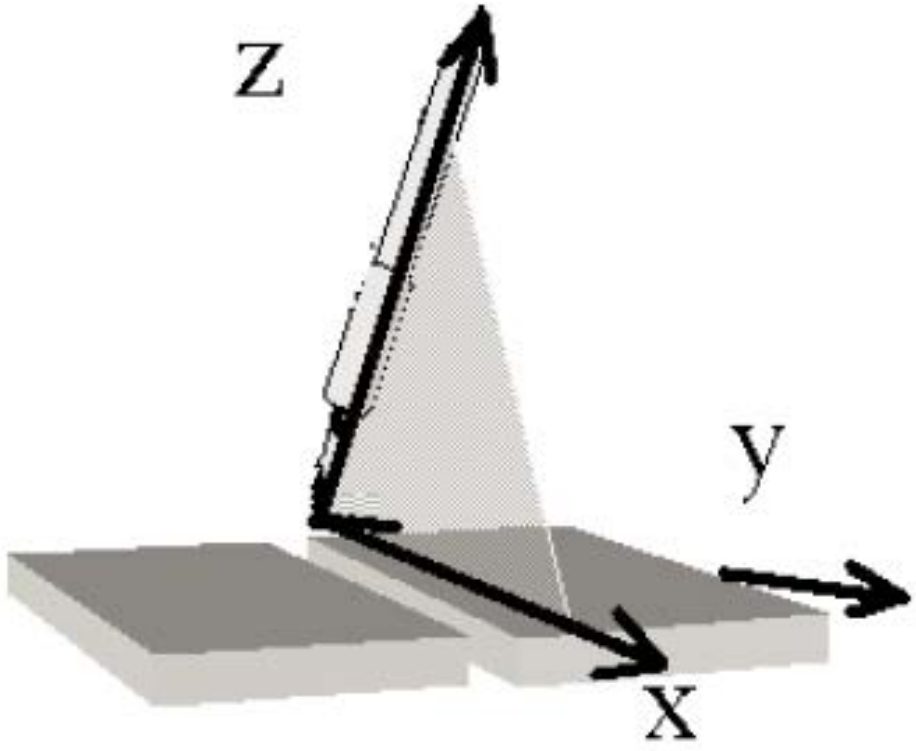
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

přeplátovaného spoje je správná pozice švu, jak je vidět na obrázku dole, ofsety švu se mohou použít ke korekci dráhy.



xx1200000199

Korekce jsou definovány v souřadném systému dráhy, který je pravostranný.



xx1200000200

- Směr x je souběžný s tečnou dráhy.
- Směr z je z-vektor nástroje.
- Směr x je kolmý na rovinu napříč směry x a z.

seamadapt_y, seamadapt_z

Datový typ: num

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.11 caplatrackdata - traťová data CAP Look-Ahead-Tracker

Continuous Application Platform (CAP)

Pokračování

Adaptační komponenty švu jsou obdobné jako ofsetové komponenty švu. Magnitudy ofsetů nejsou, nicméně, dány jako pevné hodnoty. Ofsety jsou vypočítávány jako změřená mezery švu násobená adaptačními hodnotami švu.

Komponenty se používají pro adaptaci ofsetu nástroje s ohledem na šev kvůli optimalizaci procesu pro různé velikosti mezery.

track_mode

Datový typ: `num`

S komponentem `track_mode` je možné selektivně ovlivňovat sledovací chování laserového sledovače.

Hodnota	Sledovací režim
0	Normální sledování. Korekce y a z jsou obě brány v úvahu.
1	Sledování jako kdyby korekce y odeslané laserovým sledovačem byly nulové. Korekce z jsou brány v úvahu. [i]
2	Sledování jako kdyby korekce z odeslané laserovým sledovačem byly nulové. Korekce y jsou brány v úvahu. [i]
3	Sledování, jako kdyby korekce y a z odeslané laserovým sledovačem byly nulové. [i]
4	Korekce y úplně vypnuta, tj. korekce komponentu y je nastavena na nulu před odesláním k robotu. Korekce z je brána v úvahu. [ii]
5	Korekce z úplně vypnuta, tj. korekce komponentu z je nastavena na nulu před odesláním k robotu. Korekce y je brána v úvahu. [ii]
6	Korekce y a z úplně vypnuty, tj. korekce komponentu y a z je nastavena na nulu před odesláním k robotu. Korekce y je brána v úvahu. [ii]
7	Korekce y zmizela, tj. TCP vrací ramped do naprogramovaného y-komponentu dráhy. Korekce z je aktivní.
8	Korekce z zmizela, tj. TCP vrací ramped do naprogramovaného z-komponentu dráhy. Korekce y je aktivní.
9	Korekce y a z zmizely, tj. TCP vrací ramped do naprogramované dráhy.
10	Korekce y zmizela, tj. TCP vrací ramped do naprogramovaného y-komponentu dráhy. Korekce z je aktivní.
11	Korekce z zmizela, tj. TCP vrací ramped do naprogramovaného z-komponentu dráhy. Korekce y je aktivní.
12	Korekce y a z zmizely, tj. TCP vrací ramped do naprogramované dráhy.

[i] U `track_mode` 1, 2 nebo 3 bude akumulovaná korekce z předchozí instrukce `CapL/CapC` chráněna pro y a/nebo z a postoupena dál k další instrukci `CapL/CapC`. To je případ během celé životnosti CAP procesu. Nový CAP proces nebude ovlivněn

[ii] U `track_mode` 4, 5 nebo 6 jsou údaje senzoru akumulovány, i když korekce y a/nebo z je nastavena na nulu před odesláním k robotu. To znamená, že může vzniknout 'sklon' na začátku a na konci instrukce `CapL/CapC`.

Základní příklady

```
PERS captrackdata captrack := ["laser1:",50,[1,10,1,0,0,0,0,0]]
CapL p1, v200, cd, wsd, cwd, z20, tWeldGun \Track:=captrack;
```

Syntaxe

```
< data object of caplatrackdata >
  < joint_no of num >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
< filter of num >
< calibframe_no of num >
< seamoffs_y of num >
< seamoffs_z of num >
< seamadapt_y of num >
< seamadapt_z of num >
< track_mode of num >
```

Související informace

	Popsáno v:
Datový typ `captrackdata`	*captrackdata - traťová data CAP na str 1460*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.12 capspeeddata - Rychlostní data pro CAP

Použití

`capspeeddata` se používá k definování všech dat týkajících se rychlosti pro CAP proces - je to součást `capdata` a definuje veškerá rychlostní data a procesní časy potřebné pro CAP proces:

- rychlost a jako dlouho by se tato rychlost měla používat při spuštění CAP procesu,
- prodleva pro pohyb robotu ve vztahu ke spuštění CAP procesu,
- rychlost pro CAP proces,

Rychlost je omezena činností robotu. Ta se mění podle typu robotu a dráhy pohybu.

Komponenty

fly_start

Datový typ: `num`

Nepoužito.

start

Datový typ: `num`

Rychlost (v mm/s) použitá při spuštění CAP procesu.

startspeed_time

Datový typ: `num`

Čas (v sekundách) pro běh při rychlosti `start`.

startmove_delay

Datový typ: `num`

Čas (v sekundách) prodlevy pohybu robotu ve vztahu ke spuštění CAP procesu.

main

Datový typ: `num`

Hlavní rychlost CAP procesu (mm/s).

fly_end

Datový typ: `num`

Nepoužito.

Konstrukce

```
< data object of capspeeddata >
  < fly_start of num >
  < start of num >
  < startspeed_time of num >
  < startmove_delay of num >
  < main of num >
  < fly_end of num >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

	Popsáno v:
Datový typ `capdata`	*capdata - CAP data na str 1450*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.13 captrackdata - traťová data CAP

Použití

`captrackdata` poskytuje instrukce `CapL/CapC` se všemi nezbytnými daty pro korekci dráhy s Look-Ahead- nebo At-Point-Tracker. Data jsou postoupena k instrukcím `CapL/C` s volitelným argumentem `\Track`.

Komponent `device` určuje, který typ sledovače bude použit. `captrackdata` nemohou být změněna v rámci sekvence instrukcí `CapL/CapC`. Komponent `device` je nastaven první instrukcí `CapL/C` - jestliže je odlišný ve zbývajících instrukcích `CapL/C` stejné sekvence instrukcí `CapL/CapC`, nebude mít žádný účinek.

Aby bylo možné změnit `captrackdata` pro použití v instrukci `CapL/CapC`, sekvence musí být nejprve ukončena nastavením komponentu `last_inst` na `TRUE` v `capdata`.

Jestliže `\Track` není přítomen v první instrukci `CapL/C` a všech následujících ve stejné sekvenci instrukcí `CapL/CapC`, nebude použita žádná korekce.

Komponenty

device

Senzorové zařízení

Datový typ: `string`

Definuje, ke kterému zařízení je senzor připojen pro použití v instrukcích `CapL/CapC` pro generování korekcí dráhy.

max_corr

Maximální přípustná korekce dráhy

Datový typ: `num`

Definuje maximální přípustnou korekci dráhy.

Pro dopředné sledovače Look-Ahead:

- Jestliže TCP ofset je vzhledem ke korekcím dráhy větší než `max_corr` a `\WarnMaxCorr` byl určen v `CapLATrSetup`, robot bude pokračuje ve své dráze, ale použitá korekce dráhy nepřekročí `max_corr`.
- Jestliže nebyl určen `\WarnMaxCorr`, je hlášena traťová chyba a vykonávání programu je zastaveno.

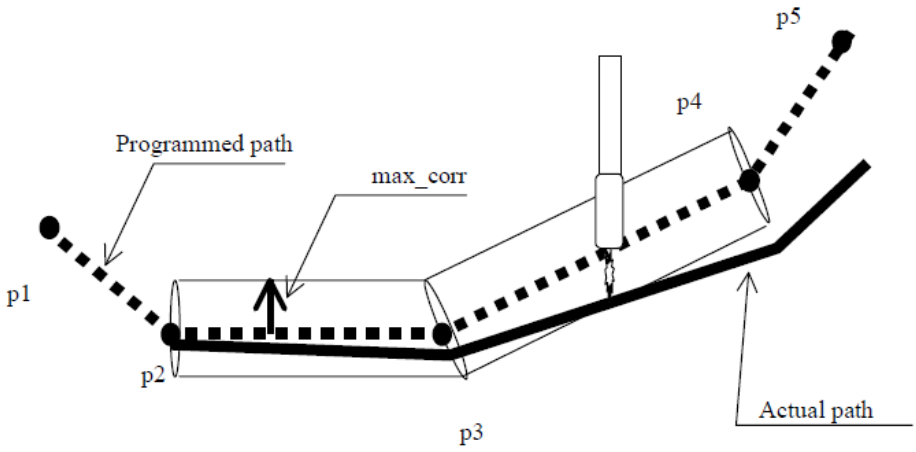
Pro sledovače At-Point:

- Jestliže TCP ofset je kvůli korekcím dráhy větší než `max_corr`, bude hlášena traťová chyba a vykonávání programu bude zastaveno.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Obrázek ukazuje nástroj v pozici vztažené k naprogramované dráze, kde je hlášena traťová chyba `max_corr`. Jednotka: mm



xx1200000198

la_trackdata

Traťová data Look-Ahead-Tracker

Datový typ: `caplatrackdata`

Definuje sledovací data, která jsou specifická pro dopředné sledovače (například laserové sledovače).

Základní příklady

SIO.cfg:

```
COM_TRP:
-Name "SCOUT:" -Type "RTP1"
-Name "digi-ip:" -Type "SOCKDEV" -PhyChannel "LAN1" -RemoteAdress
     "192.168.125.5"
```

Program RAPID:

```
PERS captrackdata captrack1 := ["digi-ip:",50,[1,10,1,0,0,0,0,0]];
CONST string laser := "digi-ip:";
PERS pose pose1 := [[137.867,-326.31,18.5],
     [0.640984,0.766438,0.0348674,0.0223137]];
PROC main()
  VAR pos sensorPos;
  CapLATrSetup laser, pose1, pos \SensorFreq:=10 \CorrFilter:=5
       \MaxBlind:=100 \MaxIncCorr:=2;
  WriteVar laser, 6, 1;
  ! sensor ON
  CapL p1, v200, cd, wsd_event, cwd, z20, tWeldGun
       \Track:=captrack1;
  CapC p2, p3, v200, cd2, wsd, cwd, z20, tWeldGun \Track:=captrack1;
  CapL p4, v200, cd3, wsd, cwd, fine, tWeldGun \Track:=captrack1;
  WriteVar laser, 6, 0;
  ! sensor OFF
ENDPROC
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.13 captrackdata - traťová data CAP

Continuous Application Platform (CAP)

Pokračování

Syntaxe

```
< data object of captrackdata >
  < device of string >
  < max_corr of num>
  < la_trackdata of caplatrackdata >
```

Související informace

	Popsáno v:
`caplatrackdata`	*caplatrackdata - traťová data CAP Look-Ahead-Tracker na str 1454*
`CapAPTrSetup`	*CapAPTrSetup - Nastavení At-Point-Tracker na str 66*
`CapLATrSetup`	*CapLATrSetup - Nastavit Look-Ahead-Tracker na str 91*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.14 capweavedata - Weavedata pro CAP

Použití

`capweavedata` se používá k definování weavingu pro CAP proces během jeho MAIN fáze (viz *Application manual - Continuous Application Platform*).

Popis weavingu

Weaving je přenesen na základní dráhu procesu. To znamená, že rychlost procesu (definovaná v `capspeeddata`) je udržována podle definice, ale rychlost TCP je zvýšena, dokud není dosaženo fyzických omezení robotu.

Dostupné typy weavingu:

- geometrický weaving: nejpřesnější tvar
- weaving zápěstí: pro weaving může být použita pouze osa 6
- rychlý weaving osy 1-3: pro weaving se používají pouze osy robotu 1, 2 a 3
- rychlý weaving osy 4-6: pro weaving se používají pouze osy robotu 4, 5 a 6

Dostupné tvary weavingu:

- Cik-cak weaving
- Weaving tvaru V
- Trojúhelníkový weaving
- Kruhový weaving

Všechny komponenty `capweavedata` se vztahují k fázi MAIN.

Komponenty

Souřadný systém dráhy je definován:

- X: směr dráhy/pohybu
- Z: z-směr nástroje
- Y: kolmý k X a Z k vystavění pravostranného souřadného systému

active

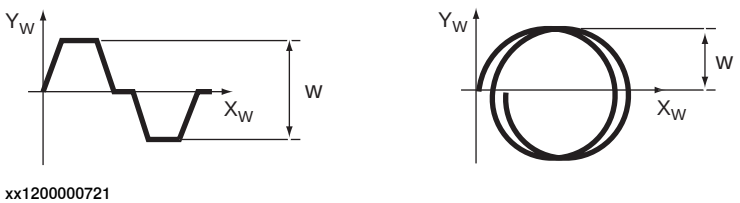
Datový typ: `bool`

Hodnota	Popis
TRUE	Provést weaving během MAIN fáze CAP procesu
FALSE	NEPROVÁDĚT weaving během MAIN fáze CAP procesu

width

Datový typ: `num`

U kruhového weavingu je šířka poloměrem kruhu (W na následujícím obrázku). U všech ostatních tvarů weavingu je šířka celkovou amplitudou obrazce weavingu.



xx1200000721

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.14 capweavedata - Weavedata pro CAP

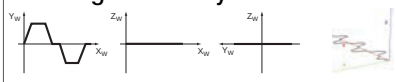
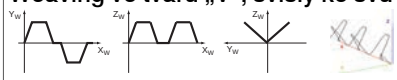
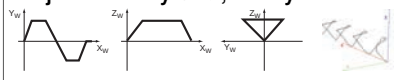
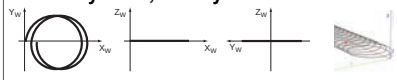
Continuous Application Platform (CAP)

Pokračování

shape

Datový typ: num

Tvar obrazce weavingu v hlavní fázi.

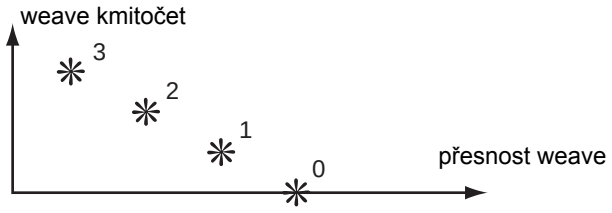
Hodnota	Geometrie tvaru	Výsledek
0	Žádný weaving	
1	Cik-cak weaving	Weaving vodorovný ke švu  xx1200000714
2	Weaving tvaru V	Weaving ve tvaru „V“, svislý ke švu  xx1200000715
3	Trojúhelníkový weaving	Trojúhelníkový tvar, svislý ke švu  xx1200000716
4	Kruhový weaving (Dostupné pouze s geometrickým weavingem, typ weavingu 0)	Ktuhový tvar, svislý ke švu  xx1200000717

type

Datový typ: num

Definuje, které osy se používají pro weaving během MAIN fáze

Určená hodnota	Typ weavingu
0	Geometrický weaving. Všechny osy se používají během weavingu.
1	Weaving zápěstí.
2	Rychlý weaving. Použity jsou osy 1, 2 a 3.
3	Rychlý weaving. Použity jsou osy 4, 5 a 6.



xx1200000718

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

POZNÁMKA

Všechny roboty, které používají TrueMove/QuickMove druhé generace, mají následující změněné chování u různých typů weavingu dostupného v RW Arc a CAP, ve srovnání s TrueMove/QuickMove první generace:

- Geometrický weaving: Beze změny.
- Weaving zápěstí: využívá hlavně osy zápěstí (4, 5 a 6), ale malé změny je možné přidat k hlavním osám, aby byly schopné udržet obrazec v požadované rovině.
- Rychlý weaving: V TrueMove/QuickMove druhé generace má geometrický weaving a weaving zápěstí vysoce zlepšený výkon. Proto již není rychlý weaving (oba typy) nezbytný jako speciální typ weavingu.

  Rychlý weaving osy 1, 2 a 3 je totožný s geometrickým weavingem.

  Rychlý weaving osy 4, 5 a 6 je totožný s weavingem zápěstí.

  Typy weavingu jsou stále dostupné pro zpětnou kompatibilitu.

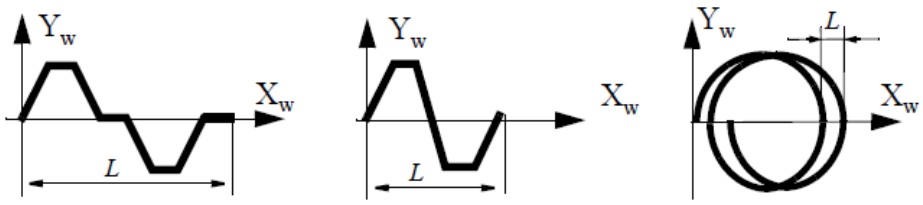
Systém používá TrueMove/QuickMove druhé generace, jestliže existuje přepínač `dyn_ipol_type 1` v MOC.cfg v datech *MOTION_PLANNER*.

length

Datový typ: `num`

Definuje délku weavingového cyklu ve fázi MAIN u geometrického weavingu (typ = 0) a weavingu zápěstí (typ = 1). Argument délky se nepoužívá pro ostatní typy weavingu.

U kruhového weavingu definuje komponent `length` vzdálenost mezi dvěma po sobě jdoucími kruhy (L), jestliže argument `cycle_time` je nastaven na 0. Jestliže `cycle_time` má hodnotu, potom může být délka posunuta. TCP se otáčí doleva s kladnou hodnotou délky a doprava se zápornou hodnotou délky.



xx1200000187

cycle_time

Datový typ: `num`

Definuje kmitočet weavingu (v Hz) ve fázi MAIN pro oba typy rychlého weavingu (typ = 2 nebo 3) a pro kruhový weaving. Argument `cycle_time` se nepoužívá u ostatních typů weavingu.

Pokračování na další straně

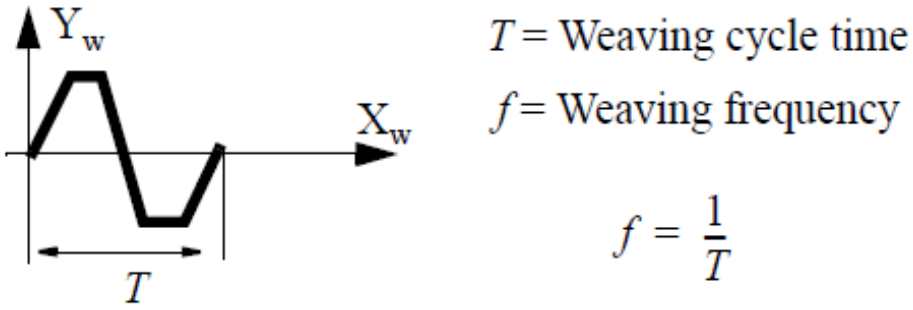
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.14 capweavedata - Weavedata pro CAP

Continuous Application Platform (CAP)

Pokračování

U kruhového weavingu definuje argument `cycle_time` počet kruhů za sekundu. TCP se otáčí doleva s kladnou hodnotou `cycle_time` a doprava se zápornou hodnotou `cycle_time`.



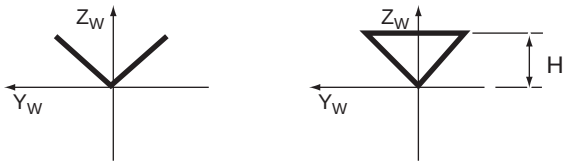
xx1200000188

height

Datový typ: `num`

Definuje výšku weavingového obrazce (v mm) během weavingu tvaru V a trojúhelníku.

Není k dispozici pro kruhový weaving.

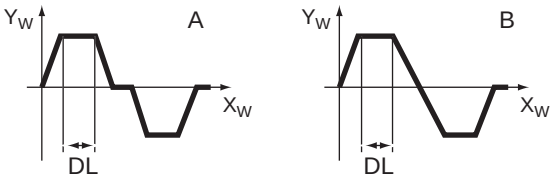


xx1200000722

dwell_left

Datový typ: `num`

Délka prodlevy (DL) použitá k přinucení TCP pohybovat se pouze ve směru švu na levém otočném bodu weave. Není k dispozici pro kruhový weaving.



xx1200000723

A	Weaving tvaru cik-cak a V.
B	Trojúhelníkový weaving

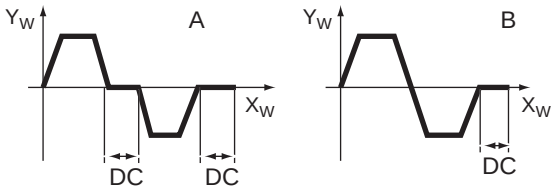
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

dwell_center

Datový typ: num

Délka prodlevy (DC) použitá k přinucení TCP pohybovat se pouze ve směru švu na středním bodu weave. Není k dispozici pro kruhový weaving.



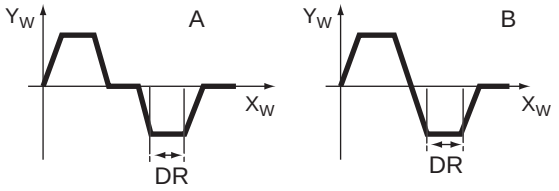
xx1200000724

A	Weaving tvaru cik-cak a V.
B	Trojúhelníkový weaving

dwell_right

Datový typ: num

Délka prodlevy (DR) použitá k přinucení TCP pohybovat se pouze ve směru švu na pravém otočném bodu weave. Není k dispozici pro kruhový weaving.



xx1200000725

A	Weaving tvaru cik-cak a V.
B	Trojúhelníkový weaving

dir

Datový typ: num

Úhel směru weave vodorovně ke švu. Výsledkem úhlu nula stupňů je weave svislý ke švu.

xx1200000726

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

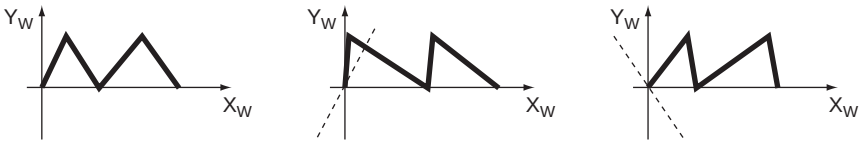
3.14 capweavedata - Weavedata pro CAP

Continuous Application Platform (CAP)
Pokračování

tilt

Datový typ: `num`

Úhel náklonu weave, svislý ke švu. Výsledkem úhlu nula stupňů je weave svislý ke švu.

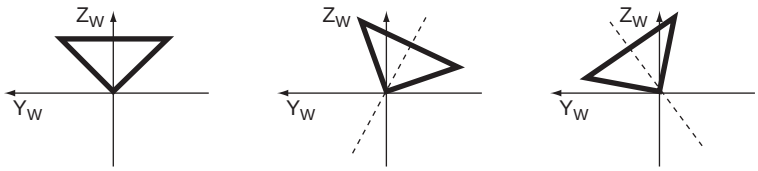


xx1200000727

rot

Datový typ: `num`

Úhel orientace weave vodorovně-svisle ke švu. Výsledkem úhlu nula stupňů je symetrický weaving.



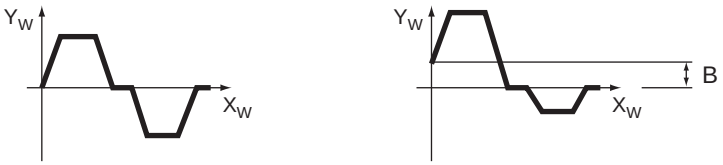
xx1200000728

bias

Datový typ: `num`

Šikmo vodorovný k weavingovému obrazci. Sklon může být určen pouze pro cik-cak weaving a nesmí být větší než polovina šířky weave. Není k dispozici u kruhového weavingu.

Následující obrázek ukazuje cik-cak weaving se sklonem a bez sklonu (B).



xx1200000729

ptrn_sync_on

Datový typ: `bool`

Hodnota	Popis
TRUE	Odeslat synchronizační impulsy na levých a pravých bodech otáčení weavingového obrazce.
FALSE	NEODESÍLAT synchronizační impulsy na levých a pravých bodech otáčení weavingového obrazce.

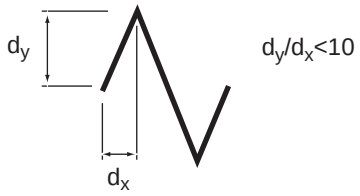
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Max kmitočet weavingu je 2 Hz.

Sklon obrazce weavingu nesmí překročit poměr 1:10 (84 stupňů). Viz následující obrázek.



xx1200000730

Změna `weave_type` v `weavedata` není možná v zónových bodech, pouze v jemných bodech. Toto je chování u spline & decbuf interpolátoru. Všechny roboty, které používají *TrueMove* or *QuickMove* druhé generace, mají následující změněné chování u různých typů weavingu dostupných v RW Arc, ve srovnání s *TrueMove* nebo *QuickMove* první generace:

- Geometrický weaving - Beze změny.
- Weaving zápěstí - využívá hlavně osy zápěstí (4, 5 a 6), ale malé změny je možné přidat k hlavním osám, aby byly schopné udržet obrazec v požadované rovině.
- Rychlý weaving: V *TrueMove* nebo *QuickMove* druhé generace má geometrický weaving a weaving zápěstí vysoce zlepšený výkon. Proto již není rychlý weaving (oba typy) nezbytný jako speciální typ weavingu.

  Rychlý weaving osy 1, 2 a 3 je totožný s geometrickým weavingem.

  Rychlý weaving osy 4, 5 a 6 je totožný s weavingem zápěstí.

  Typy weavingu jsou stále dostupné pro zpětnou kompatibilitu.

Systém používá *TrueMove* nebo *QuickMove* druhé generace, jestliže existuje přepínač `dyn_ipol_type 1` v MOC.cfg v datech MOTION_PLANNER (systémové parametry).

Syntaxe

```
< data object of capweavedata >
  < active of bool>
  < width of num >
  < shape of num >
  < type of num >
  < length of num >
  < cycle_time of num >
  < height of num >
  < dwell_left of num >
  < dwell_center of num >
  < dwell_right of num >
  < dir of num >
  < tilt of num >
  < rot of num >
  < bias of num >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.14 capweavedata - Weavedata pro CAP

Continuous Application Platform (CAP)

Pokračování

```
< ptrn_sync_on of bool >
```

Související informace

	Popsáno v:
Datový typ `capdata`	*capdata - CAP data na str 1450*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.15 cfgdomain - Konfigurační doména

Použití

`cfgdomain` (*configuration domain*) se používá k určení konfigurační domény.

Popis

Data typu `cfgdomain` jsou určena pro používání k definování konfigurační domény, která bude uložena s instrukcí `SaveCfgData`.

Základní příklady

Následující příklad názorně ukazuje datový typ `cfgdomain`:

Příklad 1

```
SaveCfgData "SYSPAR" \File:="MYEIO.cfg", EIO_DOMAIN;
```

Ukládání konfigurace I/O domény do souboru `MYEIO.cfg` v adresáři `SYSPAR`.

Předdefinovaná data

Následující předdefinované konstanty se mohou používat k určení konfigurační domény.

Název	Popis
`EIO_DOMAIN`	Systémová konfigurace I/O
`MOC_DOMAIN`	Konfigurace pohybu
`SIO_DOMAIN`	Komunikační doména
`PROC_DOMAIN`	Procesní doména
`SYS_DOMAIN`	Doména řadiče
`MMC_DOMAIN`	Komunikace člověk-stroj
`ALL_DOMAINS`	Všechny domény uvedené shora

Charakteristika

`cfgdomain` je datový typ alias pro `string` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Uložit systémové parametry do souboru	*SaveCfgData - Uložit systémové parametry do souboru na str 581*
Systémové parametry	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.16 clock - Měření času (hodiny)

Použití

`Clock` se používá pro měření času. Funkce hodin jako stopky se používají pro časování.

Popis

Data typu `clock` ukládají měření času v sekundách a mají rozlišení 0,001 sekundy.

Základní příklady

Následující příklad názorně ukazuje datový typ `clock`:

Příklad 1

```
VAR clock myclock;
ClkReset myclock;
```

Hodiny, `myclock`, jsou deklarovány a resetovány. Před použitím `ClkReset`, `ClkStart`, `ClkStop` a `ClkRead` musíte ve svém programu deklarovat proměnnou datového typu `clock`.

Omezení

Max čas, který může být uložen do proměnné hodin, je přibližně 49 dní (4 294 967 sekund). Instrukce `ClkStart`, `ClkStop` a `ClkRead` hlásí přetečení hodin při velmi nepravděpodobné události, která by mohla nastat.

Hodiny musí být deklarovány jako typ proměnné `VAR`, nikoliv jako typ proměnné `persistent`.

Charakteristika

`clock` je nehodnotový datový typ a nemůže se používat v operacích orientovaných na hodnotu.

Související informace

Pro informace o	Viz
Souhrn instrukcí k času a datumu	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Systémový čas*
Vlastnosti nehodnotového datového typu	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.17 confdata - Konfigurační data robotu

Použití

`confdata` se používá pro definování konfigurací os robotu.

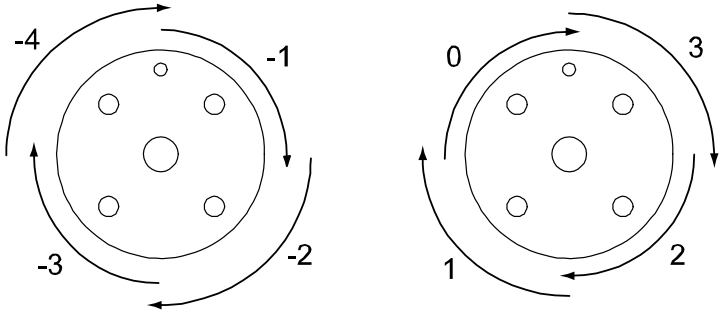
Popis

Všechny pozice robotu jsou definovány a uloženy pomocí pravoúhlých souřadnic. Při výpočtu odpovídajících pozic os budou často dvě nebo více možných řešení. To znamená, že robot je schopen dosáhnout stejné pozice, tj. nástroj je ve stejné pozici a se stejnou orientací jako několik různých pozic nebo konfigurací os robotu.

Některé typy robotů používají iterační numerické metody k určení pozic os robotu. V těchto případech se mohou použít konfigurační parametry k definování dobrých počátečních hodnot pro spoje, které budou použity iterační procedurou.

Aby bylo možné jednoznačně označit jednu z těchto možných konfigurací, je konfigurace robotu určena pomocí hodnot čtyř os. U rotační osy hodnota definuje aktuální kvadrant osy robotu. Kvadranty jsou očíslovány 0, 1, 2 a tak dále (mohou být také záporné). Číslo kvadrantu je připojeno k aktuálnímu úhlu spoje osy. U každé osy je kvadrant 0 první čtvrtinou otáčky, 0 až 90°, v kladném směru od nulové pozice; kvadrant 1 je další otáčka, 90 až 180°, a tak dále. Kvadrant -1 je otáčka 0°až (-90°) a tak dále.

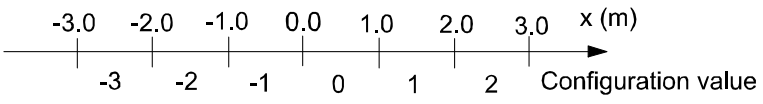
Obrázek ukazuje konfigurační kvadranty pro osu 6.



xx0500002398

U lineární osy hodnota definuje metrový interval pro osu robotu. U každé osy znamená hodnota 0 pozici mezi 0 a 1 metrem a 1 znamená pozici mezi 1 a 2 metry. U záporných hodnot znamená -1 pozici mezi -1 a 0 metrem a tak dále.

Obrázek ukazuje konfigurační hodnoty pro lineární osu.



xx0500002399

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.17 confdata - Konfigurační data robotu
RobotWare - OS
Pokračování

Dohled konfigurace

U některých modelů robotů se konfigurační data (`confdata`) používají také k provádění dohledu naprogramovaných bodů pro lineární pohyby, jestliže je nastaven `ConfL\On`.

Před spuštěním přikázaného pohybu je provedena verifikace, aby bylo vidět, jestli je možné dosáhnout naprogramované pozice. Jestliže to není možné, program je zastaven. Po dokončení pohybu (v zóně nebo v jemném bodu) je také verifikováno, že robot dosáhl naprogramované pozice.

Dohled konfigurace funguje odlišně u různých robotů. Podrobnosti najdete v následujících sekcích.

6osé roboty

Dohled konfigurace bude kontrolovat, jestli se osy 1, 4 a 6 nepohybují víc než 180 stupňů, a jestli přikázaný pohyb nevyžaduje změnu v `cfx` (`cfx` se používá pouze u robotů se sériovou linkou).

4osé roboty

Dohled konfigurace bude kontrolovat, jestli se osy 1 a 6 nepohybují víc než 180 stupňů.

Roboty se souběžným ramenem

Dohled konfigurace bude kontrolovat, jestli se osa 4 nepohybuje víc než 180 stupňů.

Roboty SCARA

Dohled konfigurace bude kontrolovat, jestli se osy 1 a 4 nepohybují víc než 180 stupňů. Bude také kontrolovat značku osy 2.

7osé roboty

Dohled konfigurace bude kontrolovat, jestli se osy 1, 4 a 6 nepohybují víc než 180 stupňů, a jestli přikázaný pohyb nevyžaduje změnu v `cfx`.

Nátěrové roboty

Není prováděn žádný dohled konfigurace.

Konfigurační data robotu

6osé roboty se sériovou linkou

V pracovním rozsahu robotu jsou tři singularity. Více informací o singularitách najdete v *Technická referenční příručka - Přehled RAPID*.

- `cf1` je číslo kvadrantu pro osu 1.
- `cf4` je číslo kvadrantu pro osu 4.
- `cf6` je číslo kvadrantu pro osu 6.

`cfx` se používá k výběru jedné z osmi možných konfigurací robotu číslovaných od 0 do 7. Následující tabulka popisuje každou z nich ve smyslu jak je robot polohován ve vztahu ke třem singularitám.

cfx	Střed zápěstí vzhledem k ose 1	Střed zápěstí vzhledem k dolnímu ramenu	Úhel osy 5
0	V přední části	V přední části	Kladný

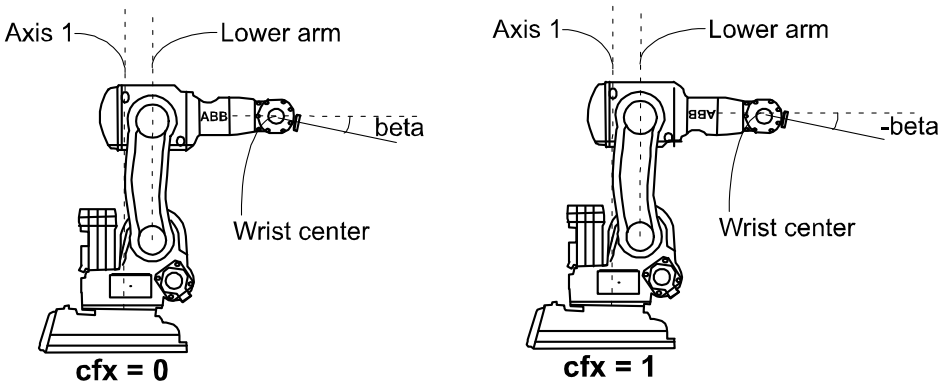
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

cfx	Střed zápěstí vzhledem k ose 1	Střed zápěstí vzhledem k dolnímu ramenu	Úhel osy 5
1	V přední části	V přední části	Záporný
2	V přední části	Za	Kladný
3	V přední části	Za	Záporný
4	Za	V přední části	Kladný
5	Za	V přední části	Záporný
6	Za	Za	Kladný
7	Za	Za	Záporný

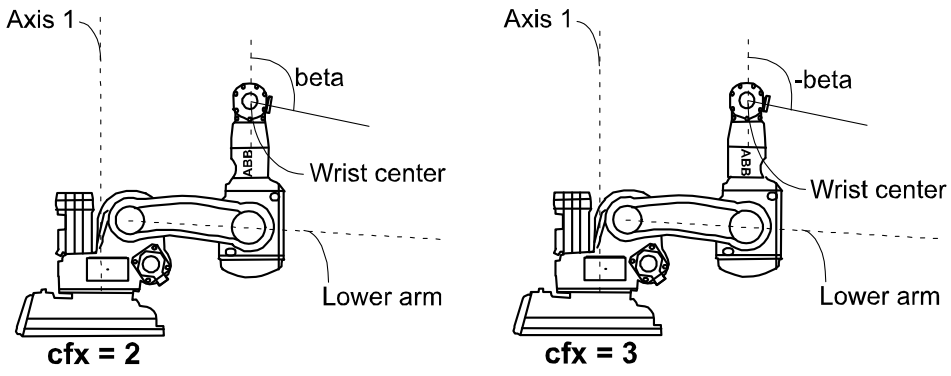
Následující obrázky popisují osm různých konfigurací se stejnou pozicí a orientací nástroje.

Následující obrázek ukazuje příklad konfigurace robotu 0 a 1. Všimněte si odlišných značek úhlu osy 5.



xx0500002400

Následující obrázek ukazuje příklad konfigurace robotu 2 a 3. Všimněte si odlišných značek úhlu osy 5.



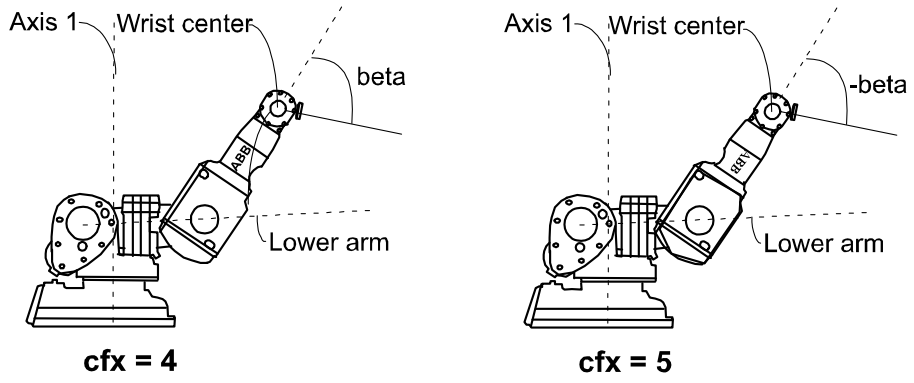
xx0500002401

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

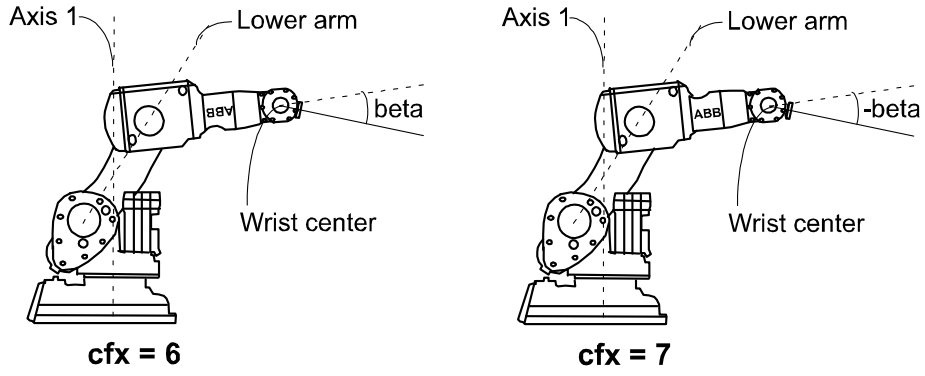
3.17 confdata - Konfigurační data robotu
RobotWare - OS
Pokračování

Následující obrázek ukazuje příklad konfigurace robotu 4 a 5. Všimněte si odlišných značek úhlu osy 5.



xx0500002402

Následující obrázek ukazuje příklad konfigurace robotu 6 a 7. Všimněte si odlišných značek úhlu osy 5.



xx0500002403

6osé roboty se souběžnou tyčí

Používají se pouze tři konfigurační parametry `cf1`, `cf4` a `cf6`.

4osé roboty

Používá se pouze konfigurační parametr `cf6`.

Roboty se souběžným ramenem

Používá se pouze konfigurační parametr `cf4`.

Roboty SCARA

Používají se pouze tři konfigurační parametry `cf1`, `cf4` a `cfx`.

Hodnota `cfx` se používá k zobrazení značky úhlu osy 2. `cfx` je 1, jestliže úhel osy 2 je záporný, jinak `cfx` je 0.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

7osé roboty

Používají se všechny čtyři konfigurační parametry. `cf1`, `cf4`, `cf6` pro spoje 1, 4 a 6. `cfx` se používá pro výběr jedné ze 16 možných konfigurací robotu.

Hodnota `cfx` je uvedena jako bitový řetězec v desítkové formě od '0000' do '1111'. Následující tabulka popisuje každý z nich ve smyslu jak je robot polohován.

`cfx`	Popis
Čtvrtý bit (1000)	Čtvrtý bit je 0, jestliže úhel osy 5 je rovný nule nebo má kladnou hodnotu. Jinak je čtvrtý bit 1.
Třetí bit (0100)	Třetí bit je 0, jestliže úhel osy 3 je rovný nebo větší než -90 stupňů. Jinak je třetí bit 1.
Druhý bit (0010)	Druhý bit je 0, jestliže úhel osy 2 je rovný nule nebo má kladnou hodnotu. Jinak je druhý bit 1.
První bit (0001)	První bit je bit kompatibility. Při programování lineárních pohybů by bit kompatibility měl být stejný jako předchozí cíl.

POZNÁMKA

Všimněte si, že vedoucí nuly nejsou zobrazeny, viz příklad dole.

Příklady konfigurace pro `cfx`:

- `cfx` = '0000' = 0

  Osa 5 = 15 stupňů, osa 3 = -55 stupňů, osa 2 = 0 stupňů, bit kompatibility = 0
- `cfx` = '0110' = 110

  Osa 5 = 15 stupňů, osa 3 = -100 stupňů, osa 2 = -1 stupňů, bit kompatibility = 0
- `cfx` = '1000' = 1000

  Osa 5 = -15 stupňů, osa 3 = 100 stupňů, osa 2 = 1 stupňů, bit kompatibility = 0

Nátěrové roboty

Používají se všechny čtyři konfigurační parametry. `cf1`, `cf4`, `cf6` pro spoje 1, 4 a 6 a `cfx` pro spoj 5.

IRB 5500

Používají se všechny čtyři konfigurační parametry. `cf1`, `cf4`, `cf6` pro svary 1, 4 a 6. Parametr `cfx` obsahuje kombinaci čísla kvadrantu svaru 5 a čtyři možné konfigurace pro osy 2 a 3.

Další informace viz *Product Manual - IRB 5500*.

IRB 5350

Robot má dvě rotační osy (ramena 1 a 2) a jednu lineární osu (rameno 3).

- `cf1` se používá pro rotační osu 1
- `cfx` se používá pro rotační osu 2
- `cf4` a `cf6` se nepoužívají

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.17 confdata - Konfigurační data robotu

RobotWare - OS

Pokračování

Komponenty

`cf1`

Datový typ: `num`

Rotační osa:

Aktuální kvadrant osy 1 vyjádřený jako kladné nebo záporné celé číslo.

Lineární osa:

Aktuální metrový interval osy 1 vyjádřený jako kladné nebo záporné celé číslo.

`cf4`

Datový typ: `num`

Rotační osa:

Aktuální kvadrant osy 4 vyjádřený jako kladné nebo záporné celé číslo.

Lineární osa:

Aktuální metrový interval osy 4 vyjádřený jako kladné nebo záporné celé číslo.

`cf6`

Datový typ: `num`

Rotační osa:

Aktuální kvadrant osy 6 vyjádřený jako kladné nebo záporné celé číslo.

Lineární osa:

Aktuální metrový interval osy 6 vyjádřený jako kladné nebo záporné celé číslo.

`cfx`

Datový typ: `num`

Rotační osa:

Pro roboty se sériovou linkou, aktuální konfigurace robotu vyjádřená jako celé číslo v rozsahu od 0 do 7.

Pro roboty SCARA, aktuální konfigurace robotu vyjádřená jako celé číslo v rozsahu od 0 do 1, viz *Roboty SCARA na str 1476*.

Pro 7osé roboty, aktuální konfigurace robotu vyjádřená jako celé číslo v rozsahu od 0 do 1111, viz *7osé roboty na str 1477*.

Pro nátěrové roboty, aktuální kvadrant osy 5 vyjádřený jako kladné nebo záporné celé číslo. Pro IRB 5500 viz *IRB 5500 na str 1477*.

Pro ostatní roboty, aktuální kvadrant osy 2 vyjádřený jako kladné nebo záporné celé číslo.

Lineární osa:

Aktuální metrový interval osy 2 vyjádřený jako kladné nebo záporné celé číslo.

Základní příklady

Následující příklad názorně ukazuje datový typ `confdata`:

Příklad 1

```
VAR confdata conf15 := [1, -1, 0, 0]
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Konfigurace robotu `conf15` pro robot nátěrového typu je definována takto:

- Konfigurace osy 1 robotu je kvadrant 1, tj. 90-180º.
- Konfigurace osy 4 robotu je kvadrant `-1`, tj. 0-(-90º).
- Konfigurace osy 6 robotu je kvadrant `0`, tj. 0 - 90º.
- Konfigurace osy 5 robotu je kvadrant `0`, tj. 0 - 90º.

Konstrukce

```
< dataobject of confdata >
  < cf1 of num >
  < cf4 of num >
  < cf6 of num >
  < cfx of num >
```

Související informace

Pro informace o	Viz
Souřadné systémy Zpracování konfiguračních dat Singularity	*Technická referenční příručka - Přehled RAPID*
Poziční data	*robtarget - Poziční data na str 1572*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.18 corrdescr - Popisovač generátoru korekcí

Použití

`corrdescr` (*Correction generator descriptor*) používají generátory korekcí. Generátor korekcí přidává geometrické ofsety do souřadného systému dráhy.

Popis

Data typu `corrdescr` obsahují odkaz na generátor korekcí.

Připojení ke generátoru korekcí se provádí instrukcí `CorrCon` a popisovač (odkaz na generátor korekcí) se může použít k dodání geometrických ofsetů do souřadného systému dráhy s instrukcí `CorrWrite`.

Ofsety poskytnuté dříve mohou být odstraněny odpojením generátoru korekcí s instrukcí `CorrDiscon`. Všechny připojené generátory korekcí mohou být odstraněny s instrukcí `CorrClear`.

Funkce `CorrRead` vrací součet všech dosud dodaných ofsetů (včetně všech připojených generátorů korekcí).

Základní příklady

Následující příklad názorně ukazuje datový typ `corrdescr`:

Příklad 1

```
VAR corrdescr id;
VAR pos offset;
...
CorrCon id;
offset := [1, 2 ,3];
CorrWrite id, offset;
```

Generátor korekcí je připojen s instrukcí `CorrCon` a odkazován popisovačem `id`. Ofsety jsou potom dodány do generátoru korekcí (s referencí `id`) pomocí instrukce `CorrWrite`.

Charakteristika

`corrdescr` je nehodnotový datový typ.

Související informace

Pro informace o	Viz
Připojuje se ke generátoru korekcí	*CorrCon - Připojuje ke generátoru korekcí na str 140*
Odpojuje se od generátoru korekcí	*CorrDiscon - Odpojuje se od generátoru korekcí na str 145*
Zapisuje do generátoru korekcí	*CorrWrite - Zapisuje do generátoru korekcí na str 146*
Načítá aktuální celkové ofsety	*CorrRead - Načítá aktuální celkové ofsety na str 1107*
CorrClear - Odstraní všechny generátory korekcí	*CorrClear - Odstraní všechny generátory korekcí na str 139*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Vlastnosti nehodnotových datových typů	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.19 datapos - Přiložený blok pro datový objekt

Použití

`datapos` je přiložený blok k datovému objektu (interní systémová data) vyhledaný funkcí `GetNextSym`.

Popis

Data typu `datapos` obsahují informace o místě, kde je určitý objekt definován v systému. Používá se pro instrukce `GetDataVal` a `SetDataVal`.

Základní příklady

Následující příklad názorně ukazuje datový typ `datapos`:

Příklad 1

```
VAR datapos block;
VAR string name;
VAR bool truevar:=TRUE;
...
SetDataSearch "bool" \Object:="my.*" \InMod:="mymod"\LocalSym;
WHILE GetNextSym(name,block) DO
  SetDataVal name\Block:=block,truevar;
ENDWHILE
```

Tato akce nastaví všechny lokální datové objekty `bool`, které začínají s `my` v modulu `mymod` na `TRUE`.

Charakteristika

`datapos` je nehodnotový datový typ.

Související informace

Pro informace o	Viz
Definovat sadu symbolů ve vyhledávací akci	*SetDataSearch - Definovat sadu symbolů ve vyhledávací sekvenci na str 622*
Získat další odpovídající symbol	*GetNextSym - Získat další odpovídající symbol na str 1175*
Získat hodnotu datového objektu.	*GetDataVal - Získat hodnotu datového objektu na str 224*
Nastavit hodnotu datového objektu	*SetDataVal - Nastavit hodnotu datového objektu na str 626*
Nastavit hodnotu mnoha objektů	*SetAllDataVal - Nastavit hodnotu všech datovým objektům v definované sadě na str 618*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.20 dionum - Digitální hodnoty (0 - 1)

Použití

`dionum`*(digital input output numeric)* se používá pro digitální hodnoty (0 nebo 1).

Tento datový typ se používá ve spojení s instrukcemi a funkcemi, které zpracovávají digitální vstupní nebo výstupní signály.

Popis

Data typu `dionum` reprezentují digitální hodnotu 0 nebo 1.

Základní příklady

Následující příklad názorně ukazuje datový typ `dionum`:

Příklad 1

```
CONST dionum close := 1;
SetDO grip1, close;
```

Definice konstanty `close` s hodnotou rovnou `1`. Signál `grip1` je potom nastaven na `close`, tj. `1`.

Předdefinovaná data

Konstanty `high`, `low` a `edge` jsou předdefinovány v systému:

```
CONST dionum low:=0;
CONST dionum high:=1;
CONST dionum edge:=2;
```

Konstanty `low` a `high` jsou určeny pro I/O instrukce.

`Edge` se může používat společně s instrukcemi přerušení `ISignalDI` a `ISignalDO`.

Charakteristika

`dionum` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Souhrn instrukcí vstup/výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*
Datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.21 dir - Struktura adresáře souborů

Použití

`dir` (*directory*) se používá pro posuv adresářových struktur.

Popis

Data typu `dir` obsahují referenci na adresář na disku nebo síti. Mohou být linkována k fyzickému adresáři prostřednictvím instrukce `OpenDir` a potom použita pro čtení.

Základní příklady

Následující příklad názorně ukazuje datový typ `dir`:

Příklad 1

```
PROC lsdir(string dirname)
  VAR dir directory;
  VAR string filename;
  OpenDir directory, dirname;
  WHILE ReadDir(directory, filename) DO
    TPWrite filename;
  ENDWHILE
  CloseDir directory;
ENDPROC
```

Tento příklad vytiskne jména všech souborů nebo podadresářů pod určeným adresářem.

Charakteristika

`dir` je nehodnotový datový typ a nemůže se používat v operacích orientovaných na hodnotu.

Související informace

Pro informace o	Viz
Otevřít adresář	*OpenDir - Otevřít adresář na str 444*
Vytvořte adresář	*MakeDir - Vytvořit nový adresář na str 338*
Načíst adresář	*ReadDir - Přečíst další vstupní data v adresáři na str 1283*
Zavřít adresář	*CloseDir - Zavřít adresář na str 125*
Odstranit adresář	*RemoveDir - Vymazat adresář na str 535*
Odstranit soubor	*RemoveFile - Vymazat soubor na str 537*
Přejmenovat soubor	*RenameFile - Přejmenovat soubor na str 540*
Zkontrolovat typ souboru	*IsFile - Zkontrolovat typ souboru na str 1207*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.22 dnum - Dvojité numerické hodnoty

Použití

`dnum` se používá pro numerické hodnoty, například počítadla. Může zpracovávat větší hodnoty celých čísel než datový typ `num`, ale jeho vlastnosti a funkce jsou stejné jako u `num`.

Popis

Hodnota datového typu `dnum` může být:

- Celé číslo, například -5
- Desetinné číslo, například 3,45

Může být také zapsáno exponenciálně, například 2E3 (= 2*10^3 = 2000), 2.5E-2 (= 0.025).

Celá čísla mezi -4503599627370496 and +4503599627370496 jsou vždy uložena jako přesná celá čísla.

Základní příklady

Následující příklady názorně ukazují datový typ `dnum`:

Příklad 1

```
VAR dnum reg1;
...
reg1:=1000000;
```

`reg1` má přidělenu hodnotu `1000000`.

Příklad 2

```
VAR dnum hex;
Var dnum bin;
VAR dnum oct;
! Hexadecimal representation of decimal value 4294967295
hex := 0xFFFFFFFF;
! Binary representation of decimal value 255
bin := 0b11111111;
! Octal representation of decimal value 255
oct := 0o377;
```

Příklad 3

```
VAR dnum a:=0;
VAR dnum b:=0;
a := 10 DIV 3;
b := 10 MOD 3;
```

Dělení celých čísel, kde pro `a` je přiděleno celé číslo (=3) a pro `b` je přidělen zbytek (=1).

Omezení

Literální hodnoty mezi -4503599627370496 až 4503599627370496 přidělené proměnné `dnum` jsou uloženy jako přesná celá čísla.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.22 dnum - Dvojité numerické hodnoty
RobotWare - OS
Pokračování

Jestliže literální hodnota, která byla interpretována jako `num`, je přidělena/použita jako `dnum`, je automaticky převedena na `dnum`.

Související informace

Pro informace o	Viz
Numerické hodnoty používající datový typ `num`	*num - Numerické hodnoty na str 1538*
Numerické výrazy	*Technická referenční příručka - Přehled RAPID*, sekce *Základní programování RAPID*
Operace pomocí numerických hodnot	*Technická referenční příručka - Přehled RAPID*, sekce *Základní programování RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.23 egmframetype - Definuje rámcové typy pro EGM

Použití

`egmframetype` se používá pro definování rámcových typů pro korekce a měření senzorů v EGM.

Popis

`egmframetype` je určeno pro používání v instrukcích `EGMActJ` a `EGMActPose`.

Základní příklady

```
CONST egm_minmax egm_minmax_lin1:=[-0.1,0.1];
CONST egm_minmax egm_minmax_rot1:=[-0.1,0.2];

EGMActPose egmID1\Tool:=tFroniusCMT\WObj:=wobj0, posecor,
     EGM_FRAME_WOBJ, posesens, EGM_FRAME_TOOL \x:=egm_minmax_lin
     \y:=egm_minmax_lin \z:=egm_minmax_lin \rx:=egm_minmax_rot
     \ry:=egm_minmax_rot \rz:=egm_minmax_rot \LpFilter:=20;
```

Předdefinované hodnoty

Hodnota	Popis
`EGM_FRAME_BASE`	Rámec je definován vzhledem k rámci základny (režim pose).
`EGM_FRAME_TOOL`	Rámec je definován vzhledem k použitému nástroji (režim pose).
`EGM_FRAME_WOBJ`	Rámec je definován vzhledem k použitému pracovnímu objektu (režim pose).
`EGM_FRAME_WORLD`	Rámec je definován vzhledem ke světovému rámci (režim joint).
`EGM_FRAME_JOINT`	Hodnoty jsou hodnotami joint (režim joint).

Charakteristika

`egmframetype` je datový typ alias pro `num`.

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.24 egmident - Identifikuje konkrétní EGM proces

Použití

`egmident` identifikuje konkrétní EGM proces.

Popis

`egmident` je rezervován pomocí instrukce `EGMGetId`. Je potom použit k identifikaci a propojení instrukcí `EGMSetupXX`, `EGMActX`, `EGMRunX` a `EGMReset` ke stejné EGM operaci.

`egmident` je identifikován podle svého jména, to znamená, druhé nebo třetí volání `EGMGetId` se stejným `egmident` nebude ani rezervovat nový proces, ani měnit jeho obsah. Pouze `EGMReset` uvolňuje `egmident`.

Základní příklady

```
VAR egmident egmID1;
VAR egmstate egmSt1;
TASK PERS wobjdata wobj_EGM1:=[FALSE, TRUE, "", [[500,700,900],
     [1,0,0,0]], [[0,0,0], [1,0,0,0]]];
CONST pose posecor:=[[1200,400,900], [0,0,1,0]];
CONST pose posesens:=[[12.3313,-0.108707,416.142],
     [0.903899,-0.00320735,0.427666,0.00765917]];
CONST egm_minmax egm_minmax_lin1:=[-0.1,0.1];
CONST egm_minmax egm_minmax_rot1:=[-0.1,0.2];
CONST egm_minmax egm_minmax_joint1:=[-0.1,0.1];

PROC testAI()
  EGMReset egmID1;
  EGMGetId egmID1;
  mvHome;
  mvHome_EGMLinear;

  egmSt1:=EGMGetState(egmID1);
  TPWrite "EGM state 1: " \Num:=egmSt1;

  IF egmSt1<=EGM_STATE_CONNECTED THEN
    EGMSetupAI ROB_1, egmID1, "default" \Pose \aiR1x:=ai_MoveX
         \aiR2y:=ai_MoveY \aiR3z:=ai_MoveZ \aiR5ry:=ai_RotY
         \aiR6rz:=ai_RotZ;
  ENDIF

  EGMActPose egmID1 \Tool:=tFroniusCMT \WObj:=wobj0, posecor,
       EGM_FRAME_WOBJ, posesens, EGM_FRAME_TOOL \x:=egm_minmax_lin1
       \y:=egm_minmax_lin1 \z:=egm_minmax_lin1 \rx:=egm_minmax_rot1
       \ry:=egm_minmax_rot1 \rz:=egm_minmax_rot1 \LpFilter:=20;
  EGMRunPose egmID1, EGM_STOP_HOLD \x \y \z \rx \ry \rz
       \RampInTime:=0.05;

  egmSt1:=EGMGetState(egmID1);
  IF egmSt1=EGM_STATE_CONNECTED THEN
    TPWrite "Reset lin 1";
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
    EGMReset egmID1;
  ENDIF
ENDPROC
```

Omezení

Existují až 4 souběžné instance, dostupné pro každou úlohu RAPID.

Charakteristika

`egmident` je nehodnotový datový typ. Nastavuje se voláním `EGMGetId`.

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.25 egm_minmax - Konvergenční kritérium pro EGM

Použití

`egm_minmax` se používá pro definování konvergenčního kritéria pro ukončení EGM.

Popis

`egm_minmax` je určeno pro používání v instrukcích `EGMActJ` a `EGMActPose`.

Komponenty

`Min`

Datový typ: `num`

Minimální odchylka

Definuje minimální hodnotu odchylky pozice. Výchozí hodnota je -0,5 stupně.

`Max`

Datový typ: `num`

Maximální odchylka

Definuje maximální hodnotu odchylky pozice. Výchozí hodnota je 0,5 stupně.

Základní příklady

```
CONST egm_minmax egm_minmax_lin1:=[-0.1,0.1];
CONST egm_minmax egm_minmax_rot1:=[-0.1,0.2];

EGMActPose egmID1\Tool:=tFroniusCMT\WObj:=wobj0, posecor,
     EGM_FRAME_WOBJ, posesens, EGM_FRAME_TOOL \x:=egm_minmax_lin1
     \y:=egm_minmax_lin1 \z:=egm_minmax_lin1 \rx:=egm_minmax_rot1
     \ry:=egm_minmax_rot1 \rz:=egm_minmax_rot1 \LpFilter:=20;
```

Charakteristika

`Egm_minmax` má následující jednotky:

- Milimetry pro x, y a z v lineárním pohybu.
- Stupně pro rx, ry, a rz v lineárním pohybu a pro společný pohyb.

Konstrukce

```
< dataobject of egm_minmax >
  < min of num >
  < max of num >
```

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.26 egmstate - Definuje stav pro EGM

Použití

`egmstate` se používá pro definování stavu pro korekce a měření senzorů v EGM.

Popis

`egmstate` je vracená hodnota funkce `EGMGetState`.

Základní příklady

```
VAR egmstate egmSt1;
VAR egmident egmID1;

EGMReset egmID1;
EGMGetId egmID1;

egmSt1:=EGMGetState(egmID1);
TPWrite "EGM state: "\Num:=egmSt1;
```

Předdefinované hodnoty

Hodnota	Popis
`EGM_STATE_DISCONNECTED`	EGM stav konkrétního procesu je nedefinován. Žádné nastavení není aktivní.
`EGM_STATE_CONNECTED`	Určený proces EGM není aktivován. Nastavení bylo provedeno, ale žádný pohyb EGM není aktivní.
`EGM_STATE_RUNNING`	Určený proces EGM běží. Pohyb EGM je aktivní, tzn. robot je v pohybu.

Charakteristika

`egmstate` je datový typ alias pro `num`.

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.27 egmstopmode - Definuje stop režimy pro EGM

Použití

`egmstopmode` se používá pro definování stop režimů pro korekce a měření senzorů v EGM.

Popis

`egmstopmode` je určeno pro používání v instrukcích `EGMRunJoint`, `EGMRunPose` a `EGMStop`.

Základní příklady

Z pohybové úlohy RAPID:

```
VAR egmstate egmSt1;
VAR egmident egmID1;

EGMReset egmID1;
EGMGetId egmID1;
CONST egm_minmax egm_minmax_lin1:=[-0.1,0.1];
CONST egm_minmax egm_minmax_rot1:=[-0.1,0.2];

EGMActPose egmID1 \Tool:=tFroniusCMT \WObj:=wobj0, posecor,
     EGM_FRAME_WOBJ, posesens, EGM_FRAME_TOOL \x:=egm_minmax_lin
     \y:=egm_minmax_lin \z:=egm_minmax_lin \rx:=egm_minmax_rot
     \ry:=egm_minmax_rot \rz:=egm_minmax_rot \LpFilter:=20;
EGMRunPose egmID1, EGM_STOP_HOLD \x \y \z \rx \ry \rz
     \RampInTime:=0.05;
```

Z RAPID TRAP nebo úlohy v pozadí:

```
EGMStop egmID1, EGM_STOP_RAMP_DOWN\RampOutTime:=5.0;
```

Předdefinované hodnoty

Hodnota	Popis
`EGM_STOP_HOLD`	Udržuje EGM koncovou pozici.
`EGM_STOP_RAMP_DOWN`	Vrací se z koncové pozice EGM do počáteční pozice.

Charakteristika

`egmstopmode` je datový typ alias pro num.

Související informace

Pro informace o	Viz
Externally Guided Motion	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.28 errdomain - Chybová doména

Použití

`errdomain` (*chybová doména*) se používá k určení chybové domény.

Popis

Data typu `errdomain` reprezentují doménu, kde je zapsána chyba, varování nebo změněný stav.

Základní příklady

Následující příklad názorně ukazuje datový typ `errdomain`:

Příklad 1

```
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
VAR trapdata err_data;
...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
ENDTRAP
```

Jestliže chyba je zachycena do trap rutiny `trap_err`, chybová doména, chybové číslo a typ chyby jsou uloženy do příslušných proměnných.

Předdefinovaná data

Následující předdefinované konstanty se mohou používat k určení chybové domény.

Název	Chybová doména	Hodnota
`COMMON_ERR`	Všechny chybové domény a domény změněného stavu	0
`OP_STATE`	Změna provozního stavu	1
`SYSTEM_ERR`	Systémové chyby	2
`HARDWARE_ERR`	Chyby hardwaru	3
`PROGRAM_ERR`	Programové chyby	4
`MOTION_ERR`	Pohybové chyby	5
`OPERATOR_ERR`	Chyby operátora - Překonáno, již se nepoužívá	6
`IO_COM_ERR`	I/O chyby a komunikační chyby	7
`USER_DEF_ERR`	Uživatelsky definované chyby (vznesené od RAPIDu)	8
`SAFETY_ERR`	Události související s bezpečností	9
`PROCESS_ERR`	Procesní chyby	11
`CFG_ERR`	Chybná konfigurace	12

Charakteristika

`errdomain` je datový typ alias pro num a následně zdědí jeho vlastnosti.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.28 errdomain - Chybová doména
RobotWare - OS
Pokračování

Související informace

Pro informace o	Viz
Přikazování přerušení na chyby	*IError - Přikazuje přerušení na chyby na str 246*
Chybová čísla	*Návod k použití - Řešení problémů, IRC5*
Datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.29 errnum - Chybové číslo

Použití

`errnum` se používá k popisu všech odstranitelných (nefatálních) chyb, které vzniknou během vykonávání programu, jako je dělení nulou.

Popis

Jestliže robot zjistí chybu během vykonávání programu, může být zpracována v chybovém handleru rutiny. Příklady takových chyb jsou hodnoty, které jsou příliš vysoké, a dělení nulou. Systémové proměnné `ERRNO`, typu `errnum` jsou tudíž přiděleny různé hodnoty podle povahy chyby. Chybový handler může být schopen opravit chybu přečtením této proměnné a potom může pokračovat vykonávání programu správným způsobem.

Chyba může být také vytvořena z programu pomocí instrukce `RAISE` . Tento konkrétní typ chyby může být zjištěn chybovým handlerem určením chybového čísla (v rozsahu 1-90 nebo zapsán s instrukcí `BookErrNo`) jako argument k `RAISE`.

Základní příklady

Následující příklady názorně ukazují datový typ `errnum`:

Příklad 1

```
reg1 := reg2 / reg3;
...
ERROR
  IF ERRNO = ERR_DIVZERO THEN
    reg3 := 1;
    RETRY;
  ENDIF
```

Jestliže `reg3 = 0`, robot zjistí chybu, když probíhá dělení. Tuto chybu, nicméně, je možné zjistit a opravit přidělením `reg3` hodnoty `1`. Potom může být dělení provedeno znovu a vykonávání programu může pokračovat.

Příklad 2

```
CONST errnum machine_error := 1;
...
IF di1=0 RAISE machine_error;
...
ERROR
  IF ERRNO=machine_error RAISE;
```

Chyba vznikne ve stroji (zjištěna prostřednictvím vstupního signálu `di1`). Je proveden skok k chybovému handleru v rutině, která zavolá chybový handler volající rutiny, kde může být chyba pravděpodobně opravena. Konstanta `machine_error` se používá k oznámení chybovému handleru, jaký druh chyby nastal.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.29 errnum - Chybové číslo
RobotWare - OS
Pokračování

Předdefinovaná data

Systémová proměnná ERRNO se může používat k přečtení poslední chyby, která se objevila. Je možné použít celou řadu předdefinovaných konstant k určení typu chyby, která nastala.

Název	Příčina chyby
ERR_ACC_TOO_LOW	Příliš malé zrychlení/zpomalení určené v instrukci PathAccLim nebo WorldAccLim
ERR_ADDR_INUSE	Adresa a port jsou již obsazeny a není možné je použít znovu. Použijte jiné číslo portu nebo jinou adresu v SocketBind.
ERR_ALIASIO_DEF	FromSignal není definován v I/O konfiguraci nebo ToSignal není deklarován v programu RAPID nebo je definován v I/O konfiguraci. Instrukce AliasIO
ERR_ALIASIO_TYPE	Signálové typy pro argumenty FromSignal a ToSignal nejsou stejné (signalx). Instrukce AliasIO.
ERR_ALRDYCNT	Proměnná přerušení je již připojena k rutině TRAP.
ERR_ALRDY_MOVING	Robot se již pohybuje při vykonávání instrukce StartMove nebo StartMoveRetry
ERR_AO_LIM	Hodnota analogového signálu je mimo limit
ERR_ARGDUPCND	Více než jeden přítomný podmíněný argument pro stejný parametr
ERR_ARGNAME	Argument je výraz, nepřítomný, nebo typu switch při vykonávání ArgName
ERR_ARGNOTPER	Argument není reference perzistentu
ERR_ARGNOTVAR	Argument není reference proměnné
ERR_ARGVALERR	Chybná hodnota argumentu
ERR_AXIS_ACT	Osa není aktivní
ERR_AXIS_IND	Osa není nezávislá
ERR_AXIS_MOVING	Osa se pohybuje
ERR_AXIS_PAR	Parametr osa v instrukci je nesprávný
ERR_BUSSTATE	IOEnable je proveden a I/O sběrnice je v chybovém stavu nebo vchází do chybového stavu před aktivací I/O jednotky.
ERR_BWDLIMIT	Omezit StepBwdPath
ERR_CALC_NEG	StrDig záporná chyba výpočtu
ERR_CALC_OVERFLOW	StrDig přetečení výpočtu
ERR_CALC_DIVZERO	StrDig dělení nulou
ERR_CALLPROC	Chyba volání procedury (není procedura) při běhu (opožděná vazba)
ERR_CAM_BUSY	Kamera je zaměstnána jiným požadavkem a nemůže provést aktuální objednávku.
ERR_CAM_COM_TIMEOUT	Komunikace proti kameře vypršela. Kamera neodpovídá.
ERR_CAM_GET_MISMATCH	Parametr získaný z kamery s instrukcí CamGetParameter má nesprávný datový typ.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Název	Příčina chyby
ERR_CAM_MAXTIME	Čas vypršel při vykonávání instrukce CamLoadJob nebo CamGetResult.
ERR_CAM_NO_MORE_DATA	Není možné získat další vizuální výsledky.
ERR_CAM_NO_PROGMODE	Kamera není v programovém režimu
ERR_CAM_NO_RUNMODE	Kamera není v režimu běhu
ERR_CAM_SET_MISMATCH	Parametr zapsaný do kamery s instrukcí CamSetParameter má špatný datový typ nebo hodnota je mimo rozsah.
ERR_CFG_INTERNAL	Není povoleno čtení interního parametru - ReadCfgData
ERR_CFG_ILL_DOMAIN	Použitá cfgdomain v instrukci SaveCfgData je neplatná nebo se nepoužívá.
ERR_CFG_ILLTYPE	Neshoda typů - ReadCfgData, WriteCfgData
ERR_CFG_LIMIT	Datový limit - WriteCfgData
ERR_CFG_NOTFND	Nenalezeno - ReadCfgData, WriteCfgData
ERR_CFG_OUTOFBOUNDS	Jestliže ListNo je -1 na vstupu nebo větší než počet dostupných instancí - ReadCfgData, WriteCfgData
ERR_CFG_WRITEFILE	Adresář neexistuje nebo použité FilePath a File jsou adresář, nebo nějaký jiný problém v souvislosti s ukládáním souboru při používání instrukce SaveCfgData.
ERR_CNTNOTVAR	Cíl CONNECT není reference proměnné
ERR_CNV_NOT_ACT	Dopravník není aktivován
ERR_CNV_CONNECT	Instrukce WaitWobj je již aktivní
ERR_CNV_DROPPED	Objekt, na který čekala instrukce WaitWobj, byl odstraněn.
ERR_COLL_STOP	Zastavení pohybu kvůli kolizi pohybu.
ERR_COMM_EXT	Komunikační chyba s externím systémem.
ERR_CONC_MAX	Počet pohybových instrukcí za sebou pomocí argumentu \Conc byl překročen.
ERR_COMM_INIT_FAILED	Komunikaèní rozhraní nelze inicializovat.
ERR_DATA_RECV	Data přijata ze vzdáleného systému jsou nesprávná.
ERR_DEV_MAXTIME	Čas vypršel při vykonávání instrukce ReadBin, ReadNum, ReadStr, ReadStrBinReadAnyBin nebo ReadRawBytes
ERR_DIPLAG_LIM	Příliš velký DipLag v instrukci TriggSpeed připojené k aktuálnímu TriggL/TriggC/TriggJ
ERR_DIVZERO	Dělení nulou
ERR_EXECPHR	Byl proveden pokus vykonat instrukci pomocí blokátoru místa
ERR_FILEACC	Nesprávný přístup k souboru
ERR_FILEEXIST	Soubor již existuje
ERR_FILEOPEN	Soubor není možné otevřít
ERR_FILNOTFND	Soubor nebyl nalezen
ERR_FNCNORET	Žádná vracená hodnota
ERR_FRAME	Není možné vypočítat nový rámec

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.29 errnum - Chybové číslo

RobotWare - OS

Pokračování

Název	Příčina chyby
ERR_GO_LIM	Hodnota digitálního analogového signálu je mimo limit
ERR_ILLDIM	Nesprávný rozměr pole
ERR_ILLQUAT	Pokus o použití ventilu s neplatnou orientací (kvaternion)
ERR_ILLRAISE	Chybové číslo v RAISE je mimo rozsah
ERR_INDCNV_ORDER	Instrukce vyžaduje vykonání IndCnvInit předtím, než je vykonána.
ERR_INOISSAFE	Jestliže zkoušíte deaktivovat dočasně bezpečné přerušení s ISleep.
ERR_INOMAX	Není k dispozici žádné další číslo přerušení
ERR_INT_NOTVAL	Neplatné celé číslo, desetinná hodnota
ERR_INT_MAXVAL	Neplatné celé číslo, příliš velká nebo malá hodnota
ERR_INVDIM	Rozměry nejsou stejné
ERR_IODISABLE	Vypršení času při vykonávání IODisable
ERR_IOENABLE	Vypršení času při vykonávání IOEnable
ERR_IOERROR	I/O chyba od instrukce Save
ERR_LINKREF	Referenční chyba v programové úloze
ERR_LOADED	Programový modul je již načten
ERR_LOADID_FATAL	Pouze interní použití v LoadId
ERR_LOADID_RETRY	Pouze interní použití v LoadId
ERR_LOADNO_INUSE	Činnost načítání se používá v StartLoad
ERR_LOADNO_NOUSE	Činnost načítání se nepoužívá v CancelLoad
ERR_MAXINTVAL	Hodnota celého čísla je příliš velká
ERR_MODULE	Nesprávné jméno modulu v instrukci Save a EraseModule
ERR_MOD_NOT_LOADED	Modul neexistuje, symbol není modul nebo jméno bylo příliš dlouhé, aby to mohl být symbol. Chyba od funkce ModTimeDnum
ERR_NAME_INVALID	Jestliže jméno I/O jednotky neexistuje
ERR_NO_ALIASIO_DEF	Proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí AliasIO.
ERR_NORUNUNIT	Jestliže není žádný kontakt s jednotkou I/O
ERR_NOTARR	Datový objekt není pole
ERR_NOTEQDIM	Použitý rozměr pole při volání rutiny se neshoduje s jejími parametry
ERR_NOTINTVAL	Nejedná se o hodnotu celého čísla
ERR_NOTPRES	Byl použit parametr navzdory faktu, že odpovídající argument nebyl použit při volání rutiny
ERR_NOTSAVED	Modul byl změněn od doby, kdy byl načten do systému
ERR_NOT_MOVETASK	Určená úloha je nepohybovou úlohou
ERR_NUM_LIMIT	Hodnota je nad 3.40282347E+38 nebo pod -3.40282347E+38

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Název	Příčina chyby
ERR_OUTOFBND	Index pole je mimo přípustné limity
ERR_OVERFLOW	Přetečení hodin
ERR_OUTSIDE_REACH	Pozice (robtarget) je mimo pracovní oblast robota pro funkci CalcJoinT.
ERR_PATH	Chybějící cílová dráha v instrukci Save
ERR_PATHDIST	Příliš dlouhá obnovovací vzdálenost pro instrukci StartMove nebo StartMoveRetry
ERR_PATH_STOP	Zastavení pohybu kvůli určité procesní chybě
ERR_PERSSUPSEARCH	Perzistentní proměnná je již TRUE na začátku procesu hledání
ERR_PID_MOVESTOP	Pouze interní použití v LoadId
ERR_PID_RAISE_PP	Chyba od ParIdRobValid, ParIdPosValid nebo LoadId.
ERR_PRGMEMFULL	Paměť programu je zaplněna
ERR_PROCSIGNAL_OFF	Procesní signál je vypnutý
ERR_PROGSTOP	Robot je ve stavu zastavení programu při vykonávání instrukce StartMove nebo StartMoveRetry
ERR_RANYBIN_CHK	Chyba kontrolního součtu byla zjištěna při přenosu dat s instrukcí ReadAnyBin
ERR_RANYBIN_EOF	Konec souboru je zjištěn před přečtením všech bajtů v instrukci ReadAnyBin
ERR_RCVDATA	Byl proveden pokus o čtení nenumerických dat s ReadNum
ERR_REFUNKDAT	Odkaz na celý neznámý datový objekt
ERR_REFUNKFUN	Odkaz na neznámou funkci
ERR_REFUNKPRC	Odkaz na neznámou proceduru v čase propojování nebo při běhu (opožděná vazba)
ERR_REFUNKTRP	Odkaz na neznámý trap
ERR_RMQ_DIM	Špatné rozměry, rozměry daných dat nejsou rovné rozměrům dat ve zprávě.
ERR_RMQ_FULL	Cílová fronta zpráv je plná.
ERR_RMQ_INVALID	Cílový slot byl ztracen nebo je neplatný
ERR_RMQ_INVMSG	Neplatná zpráva, pravděpodobně odeslána od jiného klienta než je úloha RAPID.
ERR_RMQ_MSGSIZE	Velikost zprávy je příliš velká. Zmenšete velikost zprávy.
ERR_RMQ_NAME	Dané jméno slotu není platné nebo nebylo nalezeno.
ERR_RMQ_NOMSG	Ve frontě není žádná zpráva, je to pravděpodobně výsledek výpadku napájení.
ERR_RMQ_TIMEOUT	Objevil se konec časového limitu při čekání na odpověď v RMQSendWait.
ERR_RMQ_VALUE	Syntaxe hodnoty neodpovídá datovému typu.
ERR_ROBLIMIT	Pozice je dostupná, ale nejméně jedna osa je mimo limity svaru nebo byly překročeny limity pro alespoň jeden spojený svar (funkce CalcJoinT)

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.29 errnum - Chybové číslo

RobotWare - OS

Pokračování

Název	Příčina chyby
ERR_SC_WRITE	Chyba při odesílání k externímu počítači
ERR_SIGSUPSEARCH	Signál má kladnou hodnotu již na začátku procesu hledání
ERR_SIG_NOT_VALID	Není možný přístup k I/O signálu. Důvodem může být, že I/O jednotka neběží nebo je chyba v konfiguraci (platné pouze pro aplikační sběrnici ICI).
ERR_SOCK_CLOSED	Socket je zavřen nebo není vytvořen
ERR_SOCK_TIMEOUT	Spojení nebylo navázáno během určeného času
ERR_SPEED_REFRESH_LIM	Potlačení mimo limit v SpeedRefresh
ERR_SPEEDLIM_VALUE	Rychlost použitá v instrukcích SpeedLimAxis a SpeedLimCheckPoint je příliš nízká.
ERR_STARTMOVE	Robot je ve stavu podržení při vykonávání instrukce StartMove nebo StartMoveRetry
ERR_STRTOOLNG	Řetězec je příliš dlouhý
ERR_SYM_ACCESS	Chyba přístupu ke čtení/zápisu symbolů
ERR_SYMBOL_TYPE	Datový objekt a proměnná použité v argumentu Value jsou odlišného typu. Při používání datových typů ALIAS dostanete také tuto CHYBU, i když typy mohou mít stejný základní datový typ. Instrukce GetDataVal, SetDataVal a SetAllDataVal.
ERR_SYNCMOVEOFF	Časový limit ukončen od SyncMoveOff
ERR_SYNCMOVEON	Časový limit ukončen od SyncMoveOn
ERR_SYNTAX	Chyba syntaxe v načteném modulu
ERR_TASKNAME	Jméno úlohy nebylo nalezeno v systému
ERR_TP_DIBREAK	Čtecí instrukce od FlexPendant byla přerušena digitálním vstupem
ERR_TP_DOBREAK	Čtecí instrukce od FlexPendant byla přerušena digitálním výstupem
ERR_TP_MAXTIME	Konec časového limitu při vykonávání čtecí instrukce od FlexPendant
ERR_TP_NO_CLIENT	Žádný klient pro interakci při používání čtecí instrukce od FlexPendant
ERR_TRUSTLEVEL	Nepovoleno vypnout jednotku I/O
ERR_TXTNOEXIST	Špatná tabulka nebo index ve funkci TextGet
ERR_UDPUC_COMM	Konec časového limitu komunikace u zařízení UdpUc
ERR_UI_INITVALUE	Chyba počáteční hodnoty ve funkci UINumEntry
ERR_UI_MAXMIN	Min hodnota je větší než max hodnota ve funkci UINumEntry
ERR_UI_NOTINT	Hodnota není celé číslo, když bylo určeno, že celé číslo by se mělo použít při používání UINumEntry
ERR_UISHOW_FATAL	Jiná chyba než ERR_UISHOW_FATAL v instrukci UIShow
ERR_UISHOW_FULL	Žádné místo nezbylo na FlexPendant pro jinou aplikaci při používání funkce UIShow
ERR_UNIT_PAR	Parametr Mech_unit v TestSignDefine je špatný

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Název	Příčina chyby
ERR_UNKINO	Neznámé číslo přerušení
ERR_UNKPROC	Nesprávná reference na činnost načítání v instrukci `WaitLoad`
ERR_UNLOAD	Chyba stahování v instrukci `UnLoad` nebo `WaitLoad`
ERR_WAITSYNCTASK	Časový limit ukončen od `WaitSyncTask`
ERR_WAIT_MAXTIME	Časový limit ukončen při vykonávání instrukce `WaitDI` nebo `WaitUntil`.
ERR_WHLSEARCH	Bez zastavení hledání
ERR_WOBJ_MOVING	Mechanická jednotka s pracovním objektem se pohybuje `CalcJointT`

Charakteristika

`errnum` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Obnovení po chybě	*Technická referenční příručka - Přehled RAPID*
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.30 errstr - Chybový řetězec

Použití

`errstr` se používá k psaní textu v chybových zprávách.

Základní příklady

Následující příklad názorně ukazuje datový typ `errstr`:

Příklad 1

```
VAR errstr arg:= "This is an example";

ErrLog 5100, \W, ERRSTR_TASK, ERRSTR_CONTEXT, arg, ERRSTR_EMPTY,
     ERRSTR_UNUSED;
```

Předdefinovaná data

Název	Popis
ERRSTR_EMPTY	Argument je prázdný
ERRSTR_UNUSED	Argument se nepoužívá
ERRSTR_TASK	Jméno aktuální úlohy
ERRSTR_CONTEXT	Kontext

Charakteristika

`errstr` je datový typ alias pro `string` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.31 errtype - Typ chyby

Použití

`errtype` (*error type*) se používá k určení typu chyby.

Popis

Data typu `errtype` reprezentují typ (změna stavu, varování, chyba) chybové zprávy.

Základní příklady

Následující příklad názorně ukazuje datový typ `errtype`:

Příklad 1

```
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
VAR trapdata err_data;
...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
ENDTRAP
```

Jestliže chyba je zachycena do trap rutiny `trap_err`, chybová doména, chybové číslo a typ chyby jsou uloženy do příslušných proměnných.

Předdefinovaná data

Následující předdefinované konstanty se mohou používat k určení typu chyby.

Název	Typ chyby	Hodnota
`TYPE_ALL`	Jakýkoliv typ chyby (změna stavu, varování, chyba)	0
`TYPE_STATE`	Změna stavu (provozní zpráva)	1
`TYPE_WARN`	Varování (jako je odstranitelná chyba RAPID)	2
`TYPE_ERR`	Chyba	3

Charakteristika

`errtype` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Přikazování přerušení na chyby	*IError - Přikazuje přerušení na chyby na str 246*
Chybová čísla	*Návod k použití - Řešení problémů, IRC5*
Datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.32 event_type - Typ událostní rutiny

Použití

`event_type` se používá k reprezentování aktuálního typu událostní rutiny se symbolickou konstantou.

Popis

S funkcí `EventType` je možné kontrolovat, jestli aktuální kód RAPID je vykonáván kvůli některé konkrétní systémové události nebo nikoliv.

Základní příklady

Následující příklad názorně ukazuje datový typ `event_type`:

Příklad 1

```
VAR event_type my_type;
...
my_type := EventType( );
```

Typ událostní rutiny, která je vykonána, bude uložen do proměnné `my_type`.

Předdefinovaná data

Následující konstanty typu `event_type` jsou předdefinovány:

Konstanta RAPID	Hodnota	Typ provedené události
`EVENT_NONE`	0	Žádná událost nebyla provedena
`EVENT_POWERON`	1	Událost POWER_ON
`EVENT_START`	2	Událost START
`EVENT_STOP`	3	Událost STOP
`EVENT_QSTOP`	4	Událost QSTOP
`EVENT_RESTART`	5	Událost RESTART
`EVENT_RESET`	6	Událost RESET
`EVENT_STEP`	7	Událost STEP

Charakteristika

`event_type` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Událostní rutiny všeobecně	*Technická referenční příručka - Systémové parametry*, sekce *Controller - Event Routine*.
Získat typ události	*EventType - Získat typ aktuální události uvnitř jakékoliv událostní rutiny na str 1150*
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.33 exec_level - Úroveň vykonávání

Použití

`exec_level` se používá k určení úrovně vykonávání programu.

Popis

S funkcí `ExecLevel`, je možné získat aktuální úroveň vykonávání pro kód `RAPID`, který je aktuálně vykonán.

Předdefinovaná data

Následující konstanty typu `exec_level` jsou předdefinovány:

Konstanta RAPID	Hodnota	Úroveň vykonávání
`LEVEL_NORMAL`	0	Vykonat na úrovni základny
`LEVEL_TRAP`	1	Vykonat v TRAP rutině
`LEVEL_SERVICE`	2	Vykonat v servisní rutině[i]

[i] S `LEVEL_SERVICE` znamená událostní rutina, servisní rutina (včetně Call Routine) a rutina přerušení od systémového vstupního signálu.

Charakteristika

`exec_level` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Získat aktuální úroveň vykonávání	*ExecLevel - Získat prováděcí úroveň na str 1153*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.34 extjoint - Pozice externích svarů

Použití

`extjoint` se používá pro definování pozic pomocných os, polohovačů nebo manipulátorů s pracovním kusem.

Popis

Robot může kontrolovat až šest pomocných os doplněných k jeho šesti interním osám, tj. celkem dvanáct os. Šest pomocných os je logicky označeno: a, b, c, d, e, f. Každá taková logická osa může být připojena k fyzické ose a v tomto případě je spojení definováno v systémových parametrech.

Data typu `extjoint` se používají k podržení hodnot pozic pro každou z logických os a - f.

U každé logické osy připojené k fyzické ose je pozice definována takto:

- Rotační osy - pozice je definována jako otočení ve stupních od kalibrační pozice.
- Lineární osy - pozice je definována jako vzdálenost v mm od kalibrační pozice.

Jestliže logická osa není připojena k fyzické ose, potom je použita hodnota 9E9 jako hodnota pozice, ukazující, že osa není připojena. V době vykonávání jsou poziční data každé osy kontrolována, jestli je odpovídající osa připojena nebo nikoliv. Jestliže uložená hodnota pozice nesouhlasí se skutečným připojením osy, platí následující:

- Jestliže pozice není definována v pozičních datech (hodnota je 9E9), potom bude hodnota ignorována, jestliže osa je připojena a neaktivována. Ale jestliže osa je aktivována, výsledkem bude chyba.
- Jestliže pozice je definována v pozičních datech, třebaže osa není připojena, potom bude hodnota ignorována.

Nebude proveden žádný pohyb, ale nebude generována žádná chyba pro osu s platnými pozičními daty, jestliže osa není aktivována.

Jestliže je použit ofset pomocné osy (instrukce `EOffsOn` nebo `EOffsSet`), potom jsou pozice určeny v souřadném systému `ExtOffs`.

Jestliže pomocná osa běží v nezávislém režimu a nový pohyb má být proveden robotem a jeho pomocnými osami, potom poziční data pro pomocné osy v nezávislém režimu nesmí být 9E9. Data musí být libovolnou hodnotou, která není používána systémem.

Komponenty

`eax_a`

external axis a

Datový typ: `num`

Pozice externí logické osy „a“ vyjádřená ve stupních nebo mm (podle typu osy).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
...
eax_f
```

external axis f

Datový typ: `num`

Pozice externí logické osy „f“ vyjádřená ve stupních nebo mm (podle typu osy).

Základní příklady

Následující příklad názorně ukazuje datový typ `extjoint`:

Příklad 1

```
VAR extjoint axpos10 := [ 11, 12.3, 9E9, 9E9, 9E9, 9E9] ;
```

Pozice externího polohovače `axpos10`, je definována takto:

- Pozice externí logické osy „a“ je nastavena na `11`, vyjádřeno ve stupních nebo mm (podle typu osy).
- Pozice externí logické osy „b“ je nastavena na `12.3`, vyjádřeno ve stupních nebo mm (podle typu osy).
- Osy c až f nejsou definovány.

Konstrukce

```
< dataobject of extjoint >
  < eax_a of num >
  < eax_b of num >
  < eax_c of num >
  < eax_d of num >
  < eax_e of num >
  < eax_f of num >
```

Související informace

Pro informace o	Viz
Poziční data	*robtarget - Poziční data na str 1572* *jointtarget - Data pozice svaru na str 1520*
Souřadný systém ExtOffs	*EOffsOn - Aktivuje ofset pro pomocné osy na str 197*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.35 flypointdata - Data pro letmý start/konec

Použití

`flypointdata` se používá k definování všech dat letmého startu nebo letmého konce pro CAP proces - je to součást `capdata` pro letmý start a letmý konec.

Definice

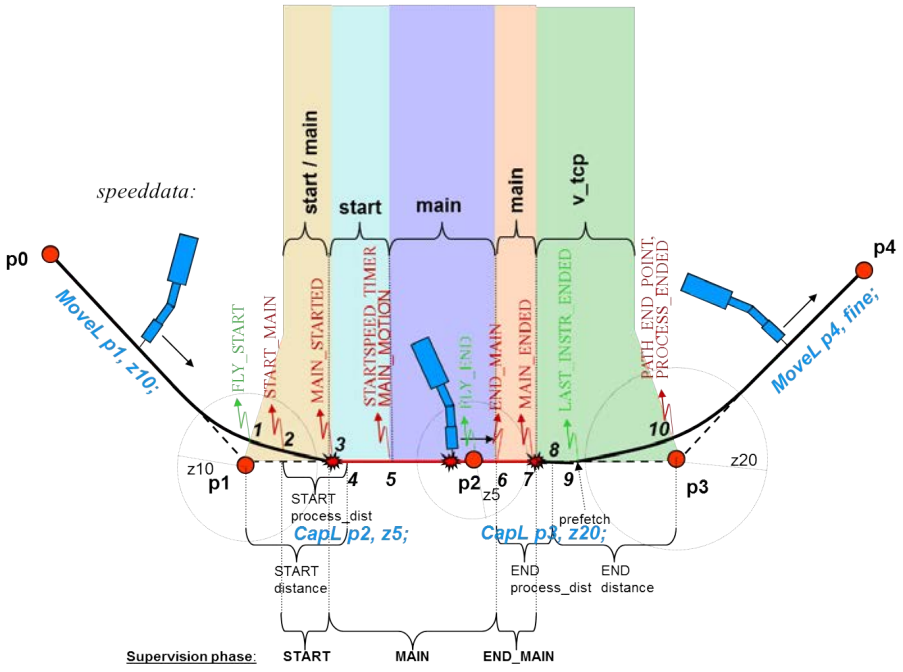
`flypointdata` definuje pro letmý start:

- Tato funkčnost je dostupná pouze pro CAP.
- Letmý start je spuštěn kombinací *první instrukce = TRUE* a zónového bodu.
- Letmý konec je spuštěn kombinací *last_instr = TRUE* a zónového bodu.
- Weavestart bude ignorován.
- Jestliže počátečním bodem je jemný bod, nebude proveden žádný letmý start.
- Jestliže konečným bodem je jemný bod, nebude proveden žádný letmý konec.
- Zpoždění pohybu bude ignorováno.
- Restart po chybě bude fungovat stejným způsobem jako obvykle: neexistují konkrétní funkce pro letmý start, je k dispozici je drhnoucí start, jestliže aplikační proces byl aktivní, když se objevila chyba.
- Jestliže je aktivován weaving, přechod do zóny je proveden rampováním ve weavingovém vzoru se začátkem na vstupu do zóny až do doby dosažení plného vzoru, když TCP opouští zónu.
- Dohled je aktivní během fáze START (s pohybujícím se TCP), fáze MAIN a fáze END_MAIN (s pohybujícím se TCP).
- Vracení se na dráze bude limitováno na vracení na pozici 4 (viz následující obrázek).
- Uživatel musí přizpůsobit vzdálenost a přiblížení a odchozí úhel aplikačnímu procesu: například u obloukového svařování v bodu, kde by měl být stanoven oblouk (bod 4 na obrázku), musí být zvolen takový způsob, kdy je možný zážeh.
- Vzdálenost mezi pozicí 4 a 6 nesmí být = 0.
- `START process_dist` musí být stejná nebo kratší než `START distance`.
- Jestliže vykonávání programu je zastaveno a aplikační proces je aktivní (mezi pozicemi 3 a 6) CAP se bude chovat jako obvykle, tzn. k dispozici je vracení na dráze (pouze když byla přejeta pos. 4), weave start, zpoždění pohybu a časový limit spuštění pohybu.
- Jestliže vykonávání programu je zastaveno mezi pozicemi 1 a 3 nebo mezi pozicemi 7 a 10, instrukce `CapX` se bude chovat jako instrukce `TrigX`.
- Doporučuje se, aby první CAP segment s letmým startem byl dlouhý alespoň `START distance`.
- Jestliže první segment je kratší než `START distance`, ale delší než `START process_dist`, pozice 2 a 4 budou posunuty k pozici 1.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

- Jestliže první segment je kratší nebo stejný jako `START process_dist`, pozice 1 a 2 se budou shodovat a pozice 4 bude na konci segmentu.
- Doporučuje se, aby první CAP segment s letmým koncem byl dlouhý alespoň `END distance + END process_dist`.
- Jestliže poslední segment je kratší než `END distance + END process_dist`, ale delší než `END process_dist`, pozice 7 a 9 budou posunuty k pozici 10.
- Jestliže poslední segment je kratší nebo stejný jako `END process_dist`, pozice 8 a 10 se budou shodovat a pozice 6 bude na začátku segmentu.
- Časový limit fáze START určený v `capdata` bude použit pouze při restartu aplikačního procesu.
- Jestliže se objeví procesní chyba poté, co požadavek přednačtení (prefetch) přišel při poslední CAP instrukci (po pozici 9), tj. PGM je uvolněno od instrukce CAP a může pokračovat s další instrukcí, je odeslána zpráva chybného zápisu, proces je zastaven, *but* pohyb robotu pokračuje.



xx1200000180

Komponenty

from_start

Datový typ: `bool`

Nepoužito.

process_dist

Datový typ: `num`

Vzdálenost (v mm), v jejímž rámci je proces spuštěn (u *letmého startu*) nebo ukončen (u *letmého konce*).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.35 flypointdata - Data pro letmý start/konec

Continuous Application Platform (CAP)
Pokračování

distance

Datový typ: `num`

Nastavuje start/konec dohledu CAP procesu jako vzdálenost (v mm) od počátečního/koncového bodu.

Konstrukce

```
< databases of flypointdata >
  < from_start of bool >
  < process_dist of num >
  < distance of num >
```

Související informace

	Popsáno v:
Datový typ `capdata`	*capdata - CAP data na str 1450*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.36 handler_type - Typ prováděcího handleru

Použití

`handler_type` se používá k určení typu prováděcího handleru v programové rutině `RAPID`.

Popis

S funkcí `ExecHandler` je možné kontrolovat, jestli aktuální kód `RAPID` je vykonáván v některém prováděcím handleru v programové rutině `RAPID`.

Základní příklady

Následující příklad názorně ukazuje datový typ `handler_type`:

Příklad 1

```
VAR handler_type my_type;
...
my_type := ExecHandler( );
```

Typ prováděcího handleru, ve kterém je vykonáván kód, bude uložen od proměnné `my_type`.

Předdefinovaná data

Následující konstanty typu `handler_type` jsou předdefinovány:

Konstanta RAPID	Hodnota	Typ prováděcího handleru
`HANDLER_NONE`	0	Není vykonáváno v žádném handleru
`HANDLER_BWD`	1	Vykonáno v handleru `BACKWARD`
`HANDLER_ERR`	2	Vykonáno v handleru `ERROR`
`HANDLER_UNDO`	3	Vykonáno v handleru `UNDO`

Charakteristika

`handler_type` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Získat typ prováděcího handleru	*ExecHandler - Získat typ prováděcího handleru na str 1152*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.37 icondata - Data ikony displeje

Použití

`icondata` se používá pro reprezentování standardních ikon na uživatelském zařízení, jako je FlexPendant.

Popis

Výčtová konstanta `icondata` může být poslána k argumentu `Icon` v instrukci `UIMsgBox` a funkcím `UIMessageBox`, `UINumEntry`, `UINumTune`, `UIAlphaEntry` a `UIListView`.

Základní příklady

Následující příklad názorně ukazuje datový typ `icondata`:

Příklad 1

```
VAR btnres answer;

UIMsgBox "More ?" \Buttons:=btnYesNo \Icon:=iconInfo \Result:=
     answer;
IF answer= resYes THEN
...
ELSEIF answer =ResNo THEN
...
ENDIF
```

Výčtová konstanta se standardním tlačítkem `iconInfo` poskytne informační ikonu v hlavičce okénka zpráv na uživatelském rozhraní.

Předdefinovaná data

Následující konstanty datového typu `icondata` jsou předdefinovány v systému:

Hodnota	Konstanta	Ikona
0	`iconNone`	Žádná ikona
1	`iconInfo`	Informační ikona
2	`iconWarning`	Varovná ikona
3	`iconError`	Chybová ikona

Charakteristika

`icondata` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Box pro zprávy uživatelské interakce	*UIMsgBox - Základní typ dialogového boxu uživatelských zpráv na str 893*
Box pro zprávy uživatelské interakce	*UIMessageBox - Pokročilý typ uživatelského boxu zpráv na str 1410*
Numerický vstup uživatelské interakce	*UINumEntry - Uživatelský číselný vstup na str 1417*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Numerické ladění uživatelské interakce	*UINumTune - Ladění uživatelských čísel na str 1423*
Alfa vstup uživatelské interakce	*UIAlphaEntry - Uživatelský vstup na str 1382*
Náhled seznamu uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.38 identno - Identita pro pohybové instrukce

Použití

`identno` (*Identity Number*) se používá pro kontrolu synchronizování dvou nebo více koordinovaných synchronizovaných pohybů mezi sebou.

Datový typ `identno` se může použít pouze v systému *MultiMove* s doplňkem *Coordinated Robots* a pouze v programových úlohách definovaných jako Motion Task (pohybová úloha).

Popis

Pohybové instrukce v systému MultiMove musí být naprogramovány s parametrem `\ID` datového typu `identno`, jestliže koordinovaný synchronizovaný pohyb a `\ID` nejsou povoleny v žádných ostatních případech.

Určené `\ID` číslo musí být stejné ve všech spolupracujících programových úlohách. Číslo id dává záruku, že pohyby se nepopletou za běhu.

V koordinovaném synchronizovaném režimu musí být stejný počet vykonaných pohybových instrukcí ve všech programových úlohách. Volitelný parametr `\ID` datového typu `identno` bude použit ke kontrole, jestli připojené pohybové instrukce jsou prováděny souběžně před spuštěním pohybů. `\ID` číslo musí být stejné v pohybových instrukcích, které jsou prováděny souběžně.

Uživatel nemusí deklarovat žádnou proměnnou typu `identno`, ale může použít číslo přímo v instrukcích (viz *Základní příklady*).

Základní příklady

Následující příklad názorně ukazuje datový typ `identno`:

Příklad 1

```
PERS tasks task_list{2} := [["T_ROB1"],["T_ROB2"]];
VAR syncident sync1;
VAR syncident sync2;

PROC proc1()
  ...
  SyncMoveOn sync1, task_list;
  MoveL *\ID:=10,v100,z50,mytool;
  MoveL *\ID:=20,v100,fine,mytool;
  SyncMoveOff sync2;
  ...
ENDPROC
```

Charakteristika

`identno` je datový typ alias pro `num` a tudíž následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Spustit koordinované synchronizované pohyby	*SyncMoveOn - Spustit koordinované synchronizované pohyby na str 758*
Ukončit koordinované synchronizované pohyby	*SyncMoveOff - Ukončit koordinované synchronizované pohyby na str 752*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.39 intnum - Identita přerušení

Použití

`intnum` (*interrupt numeric*) se používá pro identifikaci přerušení.

Popis

Když proměnná typu `intnum` je připojena k trap rutině, je jí dána konkrétní hodnota identifikující přerušení. Tato proměnná se potom použije při všech zpracováních přerušení, jako je přikazování nebo vypínání přerušení.

Více než jedna identita přerušení může být připojena ke stejné trap rutině. Systémová proměnná `INTNO` může být tudíž použita v trap rutině k určení typu přerušení, které se objeví.

Proměnná typu `intnum` musí být vždy deklarována globální v modulu.

Základní příklady

Následující příklady názorně ukazují datový typ `intnum`:

Příklad 1

```
VAR intnum feeder_error;
...
PROC main()
  CONNECT feeder_error WITH correct_feeder;
  ISignalDI di1, 1, feeder_error;
```

Přerušení je generováno, když vstup di1 je nastaven na 1. Když k tomu dojde, je provedeno volání k trap rutině correct_feeder.

Příklad 2

```
VAR intnum feeder1_error;
VAR intnum feeder2_error;
...
PROC init_interrupt()
...
  CONNECT feeder1_error WITH correct_feeder;
  ISignalDI di1, 1, feeder1_error;
  CONNECT feeder2_error WITH correct_feeder;
  ISignalDI di2, 1, feeder2_error;
...
ENDPROC
...
TRAP correct_feeder
  IF INTNO=feeder1_error THEN
  ...
  ELSE
  ...
  ENDIF
...
ENDTRAP
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Přerušení je generováno, když některý ze vstupů `di1` nebo `di2` je nastaven na `1`. Potom je provedeno volání k trap rutině `correct_feeder`. Systémová proměnná `INTNO` se použije v trap rutině kvůli zjištění, který typ přerušení se objevil.

Omezení

Max počet aktivních proměnných typu `intnum` v kterémkoliv čase (mezi `CONNECT` a `IDelete`) je omezen na 100. Max počet přerušení ve frontě pro vykonání rutiny `TRAP` v kterémkoliv čase je omezen na 30.

Charakteristika

`Intnum` je datový typ alias pro `num` a tudíž následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Přerušení*
Datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*
Připojování přerušení	*CONNECT - Připojuje přerušení k trap rutině na str 133*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.40 iodev - Sériové kanály a soubory

Použití

`iodev` *(I/O device)* se používá pro sériové kanály, jako jsou tiskárny a soubory.

Popis

Data typu `iodev` obsahují referenci na soubor nebo sériový kanál. Mohou být propojena k fyzické jednotce prostřednictvím instrukce `Open` a potom použita pro čtení a zápis.

Základní příklady

Následující příklad názorně ukazuje datový typ `iodev`:

Příklad 1

```
VAR iodev file;
...
Open "HOME:/LOGDIR/INFILE.DOC", file\Read;
input := ReadNum(file);
```

Soubor `INFILE.DOC` je otevřen pro čtení. Při čtení ze souboru je použito `file` jako reference namísto jména souboru.

Charakteristika

`iodev` je nehodnotový datový typ.

Související informace

Pro informace o	Viz
Komunikace přes sériové kanály	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Komunikace*
Konfigurace sériových kanálů	*Technická referenční příručka - Systémové parametry*
Vlastnosti nehodnotových datových typů	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.41 iounit_state - Stav I/O jednotky

Použití

`iounit_state` se používá pro zrcadlení stavu, ve kterém se I/O jednotka aktuálně nachází.

Popis

Konstanta `iounit_state` je určena k použití při kontrole vracené hodnoty od funkce `IOUnitState`.

Základní příklady

Následující příklad názorně ukazuje datový typ `iounit_state`:

Příklad 1

```
IF (IOUnitState ("UNIT1" \Phys) = IOUNIT_PHYS_STATE_RUNNING) THEN
  ! Possible to access some signal on the I/O unit
ELSE
  ! Read/Write some signal on the I/O unit result in error
ENDIF
```

Byl proveden test, jestli I/O jednotka `UNIT1` je zapnuta a běží.

Předdefinovaná data

Předdefinované symbolické konstanty datového typu `iounit_state` byly nalezeny ve funkci `IOUnitState`.

Charakteristika

`iounit_state` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Získat aktuální stav I/O jednotky	*IOUnitState - Získat aktuální stav I/O jednotky na str 1204*
Instrukce Vstup/Výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.42 jointtarget - Data pozice svaru

Použití

`jointtarget` se používá k definování pozice, ke které se robot a externí osy posunou s instrukcí `MoveAbsJ`.

Popis

`jointtarget` definuje každou pozici individuální osy, jak pro robot, tak i pro externí osy.

Komponenty

`robax`

robot axes

Datový typ: `robjoint`

Pozice os robotu ve stupních

Pozice osy je definována jako otáčení ve stupních pro příslušnou osu (rameno) v kladném nebo záporném směru od kalibrační pozice osy.

`extax`

external axes

Datový typ: `extjoint`

Pozice externích os.

Pozice `is defined` následně pro každou jednotlivou osu (`eax_a`, `eax_b ... eax_f`):

- Rotační osy - pozice je definována jako otočení ve stupních od kalibrační pozice.
- Lineární osy - pozice je definována jako vzdálenost v mm od kalibrační pozice.

Externí osy `eax_a ...` jsou logickými osami. Jaký vztah mezi sebou má číslo logické osy a číslo fyzické osy, to je definováno v systémových parametrech.

Hodnota 9E9 je definována pro osy, které nejsou připojeny. Jestliže se liší osy definované v pozičních datech od os, které jsou aktuálně připojené při vykonávání programu, nastává následující:

- Jestliže pozice není definována v pozičních datech (hodnota je 9E9), potom bude hodnota ignorována, jestliže osa je připojena a neaktivována. Ale jestliže osa je aktivována, výsledkem bude chyba.
- Jestliže pozice je definována v pozičních datech, třebaže osa není připojena, potom bude hodnota ignorována.

Nebude proveden žádný pohyb, ale nebude generována žádná chyba pro osu s platnými pozičními daty, jestliže osa není aktivována.

Jestliže některá pomocná osa běží v nezávislém režimu a některý nový pohyb má být proveden robotem a jeho pomocnými osami, potom poziční data pro externí osy v nezávislém režimu nesmí být 9E9, ale nějaká libovolná hodnota, která není používána systémem.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Základní příklady

Následující příklad názorně ukazuje datový typ `jointtarget`:

Příklad 1

```
CONST jointtarget calib_pos := [ [ 0, 0, 0, 0, 0, 0], [ 0, 9E9,
    9E9, 9E9, 9E9, 9E9] ];
```

Normální kalibrační pozice pro IRB2400 je definována v `calib_pos` datovým typem `jointtarget`. Normální kalibrační pozice 0 (stupně nebo mm) je také definována pro externí logickou osu a. Externí osy b až f nejsou definovány.

Konstrukce

```
< dataobject of jointtarget >
  < robax of robjoint >
    < rax_1 of num >
    < rax_2 of num >
    < rax_3 of num >
    < rax_4 of num >
    < rax_5 of num >
    < rax_6 of num >
  < extax of extjoint >
    < eax_a of num >
    < eax_b of num >
    < eax_c of num >
    < eax_d of num >
    < eax_e of num >
    < eax_f of num >
```

Související informace

Pro informace o	Viz
Přesunout do pozice svaru	*MoveAbsJ - Posune robot do absolutní pozice spoje na str 352* *MoveExtJ - Uvést do pohybu jednu nebo několik mechanických jednotek bez TCP na str 385*
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Konfigurace externích os	*Application manual - Additional axes and stand alone controller*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.43 listitem - Vypsat datovou strukturu položek

Použití

`listitem` se používá k definování řádek nabídky včetně textu s volitelnými malými ikonami na uživatelském zařízení, jako je FlexPendant.

Popis

Data typu `listitem` umožňují uživateli definovat řádky nabídky pro funkci `UIListView`.

Základní příklad

Následující příklad názorně ukazuje datový typ `listitem` :

Příklad 1

```
CONST listitem list {3}:=[[stEmpty, "Item1"], [stEmpty, "Item2"],
     [stEmpty, "Item3"]];
```

Nabídkový seznam s položkami Item1....Item3 pro použití ve funkci `UIListView`.

Komponenty

Datový typ má následující komponenty:

`image`

Datový typ: `string`

Dráha včetně jména souboru pro zobrazení obrázku ikony (není implementováno v tomto vydání softwaru).

Použijte prázdný řetězec `""` nebo `stEmpty`, jestliže není žádná ikona k zobrazení.

`text`

Datový typ: `string`

Text pro řádku nabídky k zobrazení.

Konstrukce

```
<dataobject of listitem>
  <image of string>
  <text of string>
```

Související informace

Pro informace o	Viz
ListView uživatelské interakce	*UIListView - Náhled uživatelského seznamu na str 1402*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.44 loaddata - Zátěžová data

Použití

`loaddata` se používá k popisu zátěží připojených k mechanickému rozhraní robotu (montážní příruba robotu),

Zátěžová data obvykle definují užitečnou zátěž nebo zatížení chapadla (nastaveno instrukcí `GripLoad` nebo `MechUnitLoad` pro polohovače) robotu, tzn. zátěž držená v chapadle robotu. `loaddata` se používá také jako součást `tooldata` k popisu zatížení nástroje.

Popis

Určené zátěže se používají k nastavení dynamického modelu robotu tak, aby pohyby robotu bylo možné řídit nejlepším možným způsobem.

VAROVÁNÍ

Je důležité vždy definovat skutečné zatížení nástroje a pokud se používá, užitečné zatížení robota (např. uchopený kus). Nesprávné definice zátěžových dat mohou vést k přetížení mechanické konstrukce robota.

Uvedení nesprávných údajů může často vést k následujícím důsledkům:

- Nebude využita maximální kapacita robota
- Bude narušena přesnost dráhy s rizikem minutí
- Vznikne riziko přetížení mechanické konstrukce

Komponenty

POZNÁMKA

V tomto popisu je `loaddata` uvedeno při používání pro užitečnou zátěž. Používání pro zatížení nástroje viz *tooldata - Data nástroje na str 1610*.

`mass`

Datový typ: `num`

Hmotnost zátěže v kg.

`cog`

center of gravity

Datový typ: `pos`

Těžiště užitečné zátěže vyjádřené v mm v souřadném systému nástroje, jestliže robot drží nástroj. Jestliže je použit stacionární nástroj, potom je těžiště pro užitečnou zátěž drženou chapadlem vyjádřeno v rámci objektu souřadného systému pracovního objektu posunutého robotem.

`aom`

axes of moment

Datový typ: `orient`

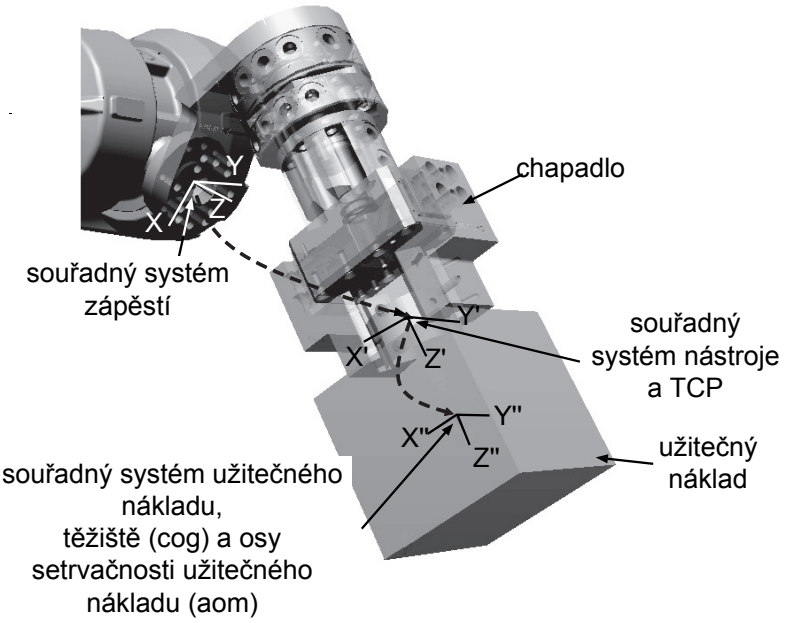
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.44 loaddata - Zátěžová data
RobotWare - OS
Pokračování

Orientace os momentu. Toto jsou hlavní osy momentu setrvačnosti užitečné zátěže s počátkem v cog**. Jestliže robot drží nástroj, osy momentu jsou vyjádřeny v souřadném systému nástroje.**

Obrázek ukazuje těžiště a setrvačné osy užitečné zátěže.



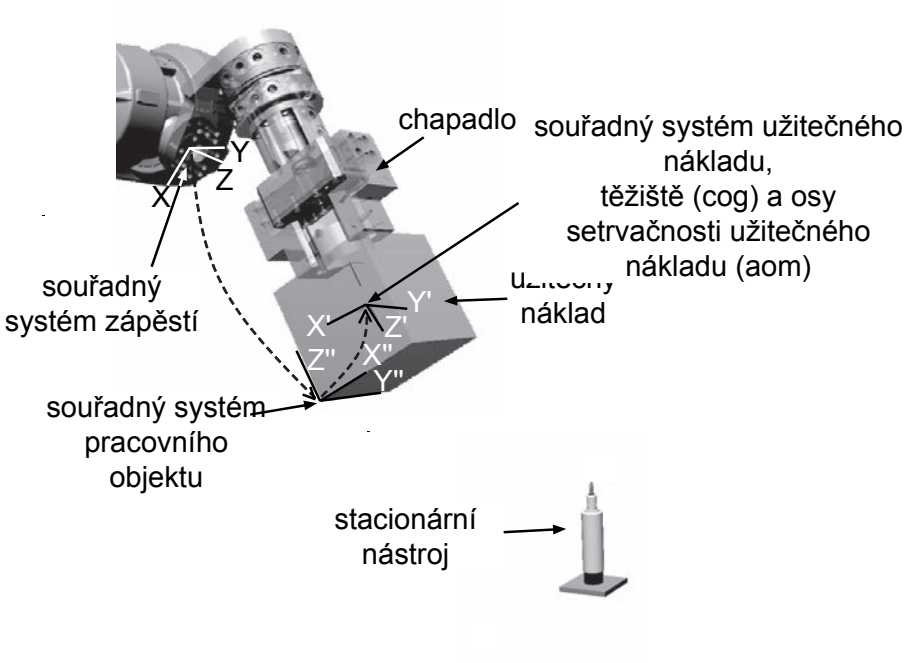
xx1100000515

Figure 3.1: Nástroj držený robotem

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Osy momentu jsou vyjádřeny v souřadném systému objektu, jestliže je použit stacionární nástroj.



xx1100000516

Figure 3.2: Stacionární nástroj

POZNÁMKA

Jestliže je použit `StationaryPayLoadMode`, osy momentu pro stacionární nástroj jsou vyjádřeny v souřadném systému zápěstí.

`ix`

inertia x

Datový typ: `num`

Moment setrvačnosti zátěže kolem osy x momentu vyjádřeného v kgm^2.

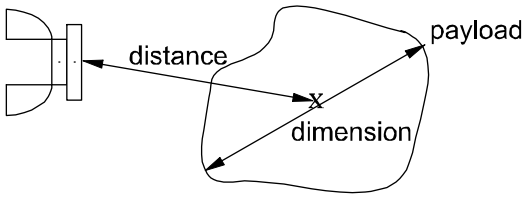
Správná definice momentů setrvačnosti umožní optimální využití plánovače dráhy a řízení os. To může být zvlášť důležité při manipulaci s velkými tabulemi plechu a tak dále. Všechny momenty setrvačnosti `ix`, `iy`, a `iz` rovného 0 kgm^2 znamenají bodovou masu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.44 loaddata - Zátěžová data
RobotWare - OS
Pokračování

Normálně musí být momenty setrvačnosti definovány, když vzdálenost od montážní příruby k těžišti je menší než max rozměr nákladu (viz následující obrázek).



xx0500002372

`iy`

inertia y

Datový typ: `num`

Moment setrvačnosti zátěže kolem osy y vyjádřený v kgm^2.

Další informace viz `ix`.

`iz`

inertia z

Datový typ: `num`

Moment setrvačnosti zátěže kolem osy z vyjádřený v kgm^2.

Další informace viz `ix`.

Základní příklady

Následující příklady názorně ukazují datový typ `loaddata`:

Příklad 1

```
PERS loaddata piece1 := [ 5, [50, 0, 50], [1, 0, 0, 0], 0, 0, 0];
```

Užitečná zátěž posunutá nástrojem drženým robotem v obrázku *Nástroj držený robotem na str 1524* je popsána pomocí následujících hodnot:

- Hmotnost `5` kg.
- Těžiště je x = `50`, y = `0` a z = `50` mm v souřadném systému nástroje
- Užitečná zátěž je bodová masa.

Příklad 2

```
Set gripper;
WaitTime 0.3;
GripLoad piece1;
```

Připojení užitečné zátěže, `piece1`, určeno ve stejný čas, když robot uchopí zátěž.

Příklad 3

```
Reset gripper;
WaitTime 0.3;
GripLoad load0;
```

Odpojení užitečné zátěže, určeno ve stejný čas, když robot odloží užitečnou zátěž.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Příklad 4

```
PERS loaddata piece2 := [ 5, [50, 50, 50], [0, 0, 1, 0], 0, 0, 0];
PERS wobjdata wobj2 :=[ TRUE, TRUE, "", [ [0, 0, 0], [1, 0, 0 ,0]
     ], [ [50, -50, 200], [0.5, 0, -0.866 ,0] ] ];
```

Užitečná zátěž posunutá podle stacionárního nástroje v obrázku *Stacionární nástroj na str 1525* je popsána pomocí následujících hodnot pro `loaddata`:

- Hmotnost 5 kg
- Těžiště je x = `50`, y = `50` a z = `50`mm v souřadném systému objektu pro pracovní objekt `wobj2`
- Souřadný systém/osy momentů užitečné zátěže jsou otočeny o 180° kolem Y“ podle souřadného systému objektu.
- Užitečná zátěž je bodová masa.

Následující hodnoty se používají pro `wobjdata`:

- Robot drží pracovní objekt
- Je použit pevný souřadný systém uživatele, tzn. souřadný systém uživatele je stejný jako souřadný systém zápěstí
- Souřadný systém objektu je otočen o -120° kolem Y a souřadnice jeho počátku jsou x = `50`, y = `-50` and z = `200` mm v souřadném systému uživatele.

Omezení

Užitečná zátěž by měla být definována pouze jako perzistentní proměnná (PERS) a nikoliv v rámci rutiny. Aktuální hodnoty jsou potom uloženy při ukládání programu a jsou vyhledány při načítání.

Argumenty typu `loaddata` v instrukci `GripLoad` a `MechUnitLoad` by měly být pouze celým perzistentem (nikoliv elementem pole nebo komponentem záznamu).

Předdefinovaná data

Zátěž `load0` definuje užitečnou zátěž s hmotností rovnou 0, tj. žádná zátěž. Tato zátěž se použije jako argument v instrukcích `GripLoad` a `MechUnitLoad` pro odpojení užitečné zátěže.

K zátěži `load0` je možné vždy přistoupit z programu, ale nemůže být změněna (je uložena v systémovém modulu BASE).

```
PERS loaddata load0 := [ 0.001, [0, 0, 0.001], [1, 0, 0, 0], 0, 0
     , 0 ];
```

Konstrukce

```
< dataobject of loaddata >
  < mass of num >
  < cog of pos >
    < x of num >
    < y of num >
    < z of num >
  < aom of orient >
    < q1 of num >
    < q2 of num >
    < q3 of num >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.44 loaddata - Zátěžová data

RobotWare - OS

Pokračování

```
  < q4 of num >
< ix of num >
< iy of num >
< iz of num >
```

Související informace

Pro informace o	Viz
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*
Definice zátěží nástroje	*tooldata - Data nástroje na str 1610*
Definovat užitečnou zátěž pro roboty	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
Definovat užitečnou zátěž pro mechanické jednotky	*MechUnitLoad - Definuje užitečnou zátěž pro mechanickou jednotku na str 343*
Zátěžová identifikace zátěže nástroje, užitečné zátěže nebo zátěže ramena	*Návod k použití - IRC5 s jednotkou FlexPendant*
Definice zátěží ramena	*Technická referenční příručka - Systémové parametry*, sekce *Téma Motion - Pracovní průběh - Jak definovat zátěž ramena*
Definice dat pracovního objektu	*wobjdata - Data pracovního objektu na str 1634*
`StationaryPayLoadMode`	*Technická referenční příručka - Systémové parametry*, část `StationaryPayLoadMode`.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.45 loadidnum - Identifikace typu zátěže

Použití

`loadidnum` se používá k zastoupení celého čísla symbolickou konstantou.

Popis

Konstanta `loadidnum` je určena k používání pro identifikaci zátěže nástroje nebo užitečné zátěže jako argumentů v instrukci `LoadId`. Viz následující příklad.

Základní příklady

Následující příklad názorně ukazuje datový typ `loadidnum`:

Příklad 1

```
! Load modules into the system
Load \Dynamic, "RELEASE:/system/mockit.sys";
Load \Dynamic, "RELEASE:/system/mockit1.sys";
%"LoadId"% TOOL_LOAD_ID, MASS_WITH_AX3, gun1;
```

Identifikace zátěže nástroje `gun1` s identifikací hmotnosti s pohyby osy 3 robotu s použitím předdefinované konstanty `MASS_WITH_AX3` datového typu `loadidnum`.

Předdefinovaná data

Následující symbolické konstanty datového typu `loadidnum` jsou předdefinovány a používají se jako argumenty v instrukci `LoadId`.

Hodnota	Symbolická konstanta	Komentář
1	`MASS_KNOWN`	Známá hmotnost v nástroji nebo užitečné zátěži.
2	`MASS_WITH_AX3`	Neznámá hmotnost v nástroji nebo užitečné zátěži. Identifikace hmotnosti bude provedena s pohyby osy 3

Charakteristika

`loadidnum` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Předdefinovaný program identifikace zátěže	*Návod k použití - IRC5 s jednotkou FlexPendant*, sekce *Programování a testování - Servisní rutiny - LoadIdentify, identifikace zátěže a servisní rutiny*
Platný typ robotu	*ParIdRobValid - Platný typ robotu pro identifikaci parametru na str 1250*
Platná pozice robotu	*ParIdPosValid - Platná pozice robotu pro identifikaci parametru na str 1247*
Identifikace zátěže s kompletním příkladem	*LoadId - Identifikace zatížení nástroje nebo užitečné zátěže na str 332*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.46 loadsession - Načtení programu

Použití

`loadsession` se používá k definování různých akcí načítání programových modulů RAPID.

Popis

Data typu `loadsession` se používají v instrukcích `StartLoad` a `WaitLoad` k identifikace akce načítání. `loadsession` obsahuje pouze referenci k akci načítání.

Charakteristika

`loadsession` je nehodnotový datový typ a nemůže se používat v operacích orientovaných na hodnotu.

Související informace

Pro informace o	Viz
Načítání programových modulů během vykonávání	*StartLoad - Načíst programový modul během vykonávání na str 703* *WaitLoad - Připojit načtený modul k úloze na str 947*
Vlastnosti nehodnotových datových typů	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.47 mecunit - Mechanická jednotka

Použití

`mecunit` se používá k definování různých mechanických jednotek, u kterých mohou být kontrola a přístup prováděny z programu.

Jména mechanických jednotek jsou definována v systémových parametrech a následně nemusí být definována v programu.

Popis

Data typu `mecunit` obsahují pouze reference k mechanické jednotce.

Omezení

Data typu `mecunit` nemusí být definována v programu. Nicméně, pokud jsou, bude zobrazena chybová zpráva, jakmile je vykonávána instrukce nebo funkce, která odkazuje k tomuto `mecunit`. Datový typ může, na druhé straně, být použit jako parametr při deklarování rutiny.

Předdefinovaná data

Všechny mechanické jednotky definované v systémových parametrech jsou předdefinovány v každé programové úloze. Ale pouze mechanické jednotky, které jsou řízeny aktuální programovou úlohou (definovanou v systémových parametrech *Controller/Task/Use Mechanical Unit Group*), jsou použity k provádění jakýchkoliv řídicích operací.

Vedle toho, předdefinovaná proměnná `ROB_ID` datového typu `mecunit` je dostupná v každé programové úloze. Jestliže aktuální programová úloha řídí robot, potom alias proměnná `ROB_ID` obsahuje reference k jednomu z robotů `ROB_1` až `ROB_6`, což může být použito k provedení řídicí operace na robotu. Proměnná `ROB_ID` je neplatná, jestliže aktuální programová úloha neřídí žádný robot.

Základní příklady

Následující příklad názorně ukazuje datový typ `mecunit`:

Příklad 1

```
IF TaskRunRob() THEN
  IndReset ROB_ID, 6;
ENDIF
```

Jestliže aktuální programová úloha řídí robot, resetujte osu `6` pro robot.

Charakteristika

`mecunit` je *nehodnotový* datový typ. To znamená, že data tohoto typu nedovolují operace orientované na hodnotu.

Související informace

Pro informace o	Viz
Zkontrolujte, jestli úloha řídí nějaký robot	*TaskRunRob - Zkontrolujte, jestli úloha řídí nějaký robot na str 1358*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.47 mecunit - Mechanická jednotka

RobotWare - OS

Pokračování

Pro informace o	Viz
Zkontrolovat, jestli úloha provozuje nějakou mechanickou jednotku	*TaskRunMec - Zkontrolujte, jestli úloha řídí nějakou mechanickou jednotku na str 1357*
Získat jména mechanických jednotek v systému	*GetNextMechUnit - Získat jméno a data pro mechanické jednotky na str 1172*
Aktivace/deaktivace mechanických jednotek	*ActUnit - Aktivuje mechanickou jednotku na str 24* *DeactUnit - Deaktivuje mechanickou jednotku na str 150*
Konfigurace mechanických jednotek	*Technická referenční příručka - Systémové parametry*
Vlastnosti nehodnotových datových typů	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.48 motsetdata - Data nastavení pohybu

Použití

`motsetdata` se používá k definování počtu nastavení pohybu, která ovlivňují všechny polohovací instrukce v programu:

- Max. rychlost a potlačení rychlosti
- Data zrychlení
- Chování kolem singulárních bodů
- Správa různých robotických konfigurací
- Potlačení rozlišení dráhy
- Monitorování pohybu
- Omezení zrychlení/zpomalení
- Reorientace nástroje během kruhové dráhy
- Aktivace a deaktivace zásobníku událostí

Tento datový typ nemusí být normálně použit, jelikož tato nastavení mohou být nastavena pouze pomocí instrukcí `VelSet`, `AccSet`, `SingArea`, `ConfJ`, `ConfL`, `PathResol`, `MotionSup`, `PathAccLim`, `CirPathMode`, `WorldAccLim`, `ActEventBuffer` a `DeactEventBuffer`.

K aktuálním hodnotám těchto nastavení pohybu se přistupuje pomocí systémové proměnné `C_MOTSET`.

Popis

Aktuální nastavení pohybu (uložené v systémové proměnné `C_MOTSET`) ovlivňují všechny pohyby.

Komponenty

`vel.oride`

Datový typ: `veldata/num`

Rychlost jako procento naprogramované rychlosti.

`vel.max`

Datový typ: `veldata/num`

Maximální rychlost v mm/s.

`acc.acc`

Datový typ: `accdata/num`

Zrychlení a zpomalení jako procento normálních hodnot.

`acc.ramp`

Datový typ: `accdata/num`

Rychlost, kterou narůstá zrychlení a zpomalení jako procento normálních hodnot.

`acc.finepramp`

Datový typ: `accdata/num`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.48 motsetdata - Data nastavení pohybu
RobotWare - OS
Pokračování

Rychlost, kterou klesá zpomalení jako procento normálních hodnot, když robot zpomaluje směrem k jemnému bodu.

`sing.wrist`

Datový typ: `singdata/bool`

Orientaci nástroje je povolena určitá odchylka kvůli ochraně před singularitou zápěstí.

`sing.lockaxis4`

Datový typ: `singdata/bool`

Zablokujte osu 4 na šestiosém robotu na nule nebo +- 180 stupních, aby se vyloučily problémy se singularitou, když osa 5 je blízko nuly.

`sing.arm`

Datový typ: `singdata/bool`

Orientaci nástroje je povolena určitá odchylka kvůli ochraně před singularitou ramena (není implementováno).

`sing.base`

Datový typ: `singdata/bool`

Orientaci nástroje není povolena odchylka.

`conf.jsup`

Datový typ: `confsupdata/bool`

Dohled nad společnou konfigurací je aktivní během společného pohybu.

`conf.lsup`

Datový typ: `confsupdata/bool`

Dohled nad společnou konfigurací je aktivní během lineárního a kruhového pohybu.

`conf.ax1`

Datový typ: `confsupdata/num`

Max přípustná odchylka ve stupních pro osu 1 (není použito v této verzi).

`conf.ax4`

Datový typ: `confsupdata/num`

Max přípustná odchylka ve stupních pro osu 4 (není použito v této verzi).

`conf.ax6`

Datový typ: `confsupdata/num`

Max přípustná odchylka ve stupních pro osu 6 (není použito v této verzi).

`pathresol`

Datový typ: `num`

Aktuální potlačení konfigurovaného rozlišení dráhy v procentech.

`motionsup`

Datový typ: `bool`

Zrcadlit stav RAPID (`TRUE` = Zapnuto a `FALSE` = Vypnuto) funkce dohledu pohybu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`tunevalue`

Datový typ: `num`

Aktuální potlačení RAPID jako procento konfigurované tunevalue pro funkci dohledu pohybu.

`acclim`

Datový typ: `bool`

Omezení zrychlení nástroje podél dráhy. (`TRUE` **= omezení a** `FALSE` **= bez omezení).**

`accmax`

Datový typ: `num`

Omezení zrychlení TCP v m/s^2. Jestliže acclim je `FALSE`**, hodnota se vždy nastavuje na -1.**

`decellim`

Datový typ: `bool`

Omezení zpomalení nástroje podél dráhy. (`TRUE` **= omezení a** `FALSE` **= bez omezení).**

`decelmax`

Datový typ: `num`

Omezení zpomalení TCP v m/s^2. Jestliže `decellim` **je** `FALSE`**, hodnota se vždy nastavuje na -1.**

`cirpathreori`

Datový typ: `num`

Reorientace nástroje během kruhové dráhy:

0 = Standardní metoda s interpolací v rámci dráhy

1 = Upravená metoda s interpolací v rámci objektu

2 = Upravená metoda s naprogramovanou orientací nástroje v `CirPoint`

`worldacclim`

Datový typ: `bool`

Omezení zrychlení ve světovém souřadném systému. (`TRUE` **= omezení a** `FALSE` **= bez omezení).**

`worldaccmax`

Datový typ: `num`

Omezení zrychlení ve světovém souřadném systému v m/s^2. Jestliže `worldacclim` **je FALSE, hodnota se vždy nastavuje na -1.**

`evtbufferact`

Datový typ: `bool`

Zásobník událostí je aktivní nebo neaktivní. (`TRUE` **= zásobník událostí je aktivní a** `FALSE` **= zásobník událostí je neaktivní).**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.48 motsetdata - Data nastavení pohybu
RobotWare - OS
Pokračování

Omezení

Jeden a pouze jeden z komponentů `sing.wrist`, `sing.arm` nebo `sing.base` mohou mít hodnotu rovnou TRUE.

Základní příklady

Následující příklad názorně ukazuje datový typ `motsetdata`:

Příklad 1

```
IF C_MOTSET.vel.oride > 50 THEN
  ...
ELSE
  ...
ENDIF
```

Různé části programu jsou vykonávány podle aktuálního potlačení rychlosti.

Předdefinovaná data

`C_MOTSET` popisuje aktuální nastavení pohybu robotu a může se k němu vždy přistupovat z programu. Na druhé straně, `C_MOTSET` je možné změnit pouze pomocí řady instrukcí, nikoliv přidělením.

Následující výchozí hodnoty pro pohybové parametry jsou nastaveny

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

```
VAR motsetdata C_MOTSET := [
  [ 100, 5000 ],-> veldata
  [ 100, 100, 100 ],-> accdata
  [ FALSE, FALSE, FALSE, TRUE ],-> singdata
  [ TRUE, TRUE, 30, 45, 90 ]-> confsupdata
  100,-> path resolution
  TRUE,-> motionsup
  100,-> tunevalue
  FALSE,-> acclim
  -1,-> accmax
  FALSE,-> decellim
  -1,-> decelmax
  0,-> cirpathreori
  FALSE,-> worldacclim
  -1,-> worldaccmax
  TRUE];-> ActEventBuffer and DeactEventBuffer
```

Konstrukce

```
<dataobject of motsetdata>
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`<vel of veldata>`
`  <oride of num>`
`  <max of num>`
	Ovlivněno instrukcí `VelSet`

`<acc of accdata>`
`  <acc of num>`
`  <ramp of num>`
`  <finepramp of num>`
	Ovlivněno instrukcí `AccSet`

`<sing of singdata>`
`  <wrist of bool>`
`  <lockaxis4 of bool>`
`  <arm of bool>`
`  <base of bool>`
	Ovlivněno instrukcí `SingArea`

`<conf of confsupdata>`
`  <jsup of bool>`
`  <lsup of bool>`
`  <ax1 of num>`
`  <ax4 of num>`
`  <ax6 of num>`
	Ovlivněno instrukcemi `ConfJ` a `ConfL`

`<pathresol of num>`	Ovlivněno instrukcí `PathResol`
`<motionsup of bool>`	Ovlivněno instrukcí `MotionSup`
`<tunevalue of num>`	Ovlivněno instrukcí `MotionSup`
`<acclim of bool>`	Ovlivněno instrukcí `PathAccLim`
`<accmax of num>`	Ovlivněno instrukcí `PathAccLim`
`<decellim of bool>`	Ovlivněno instrukcí `PathAccLim`
`<decelmax of num>`	Ovlivněno instrukcí `PathAccLim`
`<cirpathreori of num>`	Ovlivněno instrukcí `CirPathMode`
`<worldacclim of bool>`	Ovlivněno instrukcí `WorldAccLim`
`<worldaccmax of num>`	Ovlivněno instrukcí `WorldAccLim`
`<evtbufferact of bool>`	Ovlivněno instrukcemi `ActEventBuffer` a `DeactEventBuffer`

Související informace

Pro informace o	Viz
Instrukce pro nastavení pohybových parametrů	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Nastavení pohybu*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.49 num - Numerické hodnoty

Použití

`Num` se používá pro numerické hodnoty; např. počítadla.

Popis

Hodnota datového typu `num` může být

- celé číslo; např. -5,
- desetinné číslo; např. 3,45

Může být také zapsáno exponenciálně, například 2E3 (= 2*10^3 = 2000), 2.5E-2 (= 0.025).

Celá čísla mezi -8388607 a +8388608 jsou vždy uložena jako přesná celá čísla.

Desetinná čísla jsou pouze přibližná čísla a proto by se neměla používat ve srovnáváních *je rovno* nebo *není rovno*. V případě dělení a operací používajících desetinná čísla bude výsledkem také desetinné číslo; tj. nikoliv přesné celé číslo. Například:

```
a := 10;
b := 5;
IF a/b=2 THEN

...
```

Výsledkem a/b není celé číslo, tato podmínka není nezbytně splněna.

Základní příklady

Následující příklady názorně ukazují datový typ `num`:

Příklad 1

```
VAR num reg1;
...
reg1 := 3;
```

`reg1` má přidělenu hodnotu `3`.

Příklad 2

```
a := 10 DIV 3;
b := 10 MOD 3;
```

Dělení celých čísel, kde pro `a` je přiděleno celé číslo (=3) a pro `b` je přidělen zbytek (=1).

Předdefinovaná data

V systému jsou některá předdefinovaná data. Například, konstanta pí (π) je definována.

CONST num pi := 3.1415926;

Omezení

Literální hodnoty mezi -8388607 až 8388608 přidělené proměnné `num` jsou uloženy jako přesná celá čísla.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Jestliže literál, který byl interpretován jako `dnum`, je přidělen/použit jako `num`, je automaticky převeden na `num`.

Související informace

Pro informace o	Viz
Numerické hodnoty používající datový typ dnum	*dnum - Dvojité numerické hodnoty na str 1485*
Numerické výrazy	*Technická referenční příručka - Přehled RAPID*, sekce *Základní programování RAPID - Výrazy*
Operace pomocí numerických hodnot	*Technická referenční příručka - Přehled RAPID*, sekce *Základní programování RAPID - Výrazy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.50 opcalc - Arithmetic Operator

Použití

`opcalc` se používá k zastoupení aritmetického operátoru v argumentech k funkcím nebo instrukcím RAPID.

Popis

Konstanta `opcalc` je určena k používání při definování typu aritmetické operace.

Příklady

Následující příklad názorně ukazuje datový typ `opcalc`:

Příklad 1

```
res := StrDigCalc(str1, OpAdd, str2);
```

Pro `res` je přidělen výsledek operace sčítání na hodnotách reprezentovaných řetězci `str1` a `str2`. `OpAdd` je datového typu `opcalc`.

Předdefinovaná data

Následující symbolické konstanty datového typu `opcalc` jsou předdefinovány a používají se k definování typu aritmetické operace, použité např. ve funkci `StrDigCalc`.

Konstanta	Hodnota	Komentář
OpAdd	1	Sčítání (+)
OpSub	2	Odčítání (-)
OpMult	3	Násobení (*)
OpDiv	4	Dělení (/)
OpMod	5	Absolutní hodnota (Modulus) (%I)

Charakteristika

`opcalc` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*
Aritmetické operace na digitálních řetězcích.	*StrDigCalc - Aritmetické operace s datovým typem stringdig na str 1333*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.51 opnum - Srovnávací operátor

Použití

`opnum` se používá k zastoupení operátoru pro srovnávání v argumentech k funkcím nebo instrukcím RAPID.

Popis

Konstanta `opnum` je určena k používání při definování typu srovnávání při kontrole hodnot v generických instrukcích.

Základní příklady

Následující příklad názorně ukazuje datový typ `opnum`:

Příklad 1

```
TriggCheckIO checkgrip, 100, airok, EQ, 1, intno1;
```

Předdefinovaná data

Následující symbolické konstanty datového typu `opnum` jsou předdefinovány a používají se k definování typu srovnávání, použitého např. v instrukci `TriggCheckIO`.

Hodnota	Symbolická konstanta	Komentář
1	`LT`	Méně než
2	`LTEQ`	Méně než nebo stejně jako
3	`EQ`	Stejně jako (rovná se)
4	`NOTEQ`	Není stejný jako
5	`GTEQ`	Větší než nebo stejný jako
6	`GT`	Větší než

Charakteristika

`opnum` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*
Definovat kontrolu I/O na pevné pozici	*TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.52 orient - Orientace

Použití

`orient` se používá pro orientace (jako je orientace nástroje) a rotace (jako je rotace souřadného systému).

Popis

Orientace je popisována formou kvaternionu, který se skládá že čtyř komponentů: `q1`, `q2`, `q3` a `q4`.

Komponenty

Datový typ `orient` má následující komponenty:

`q1`

Datový typ: `num`

Kvaternion 1.

`q2`

Datový typ: `num`

Kvaternion 2.

`q3`

Datový typ: `num`

Kvaternion 3.

`q4`

Datový typ: `num`

Kvaternion 4.

Základní příklady

Následující příklad názorně ukazuje datový typ `orient`:

Příklad 1

```
VAR orient orient1;
.
orient1 := [1, 0, 0, 0];
```

Pro orientaci `orient1` je přidělena hodnota q1=1, q2-q4=0; to odpovídá žádné rotaci.

Omezení

Orientace musí být normalizována; tj. součet čtverců se musí rovnat 1:

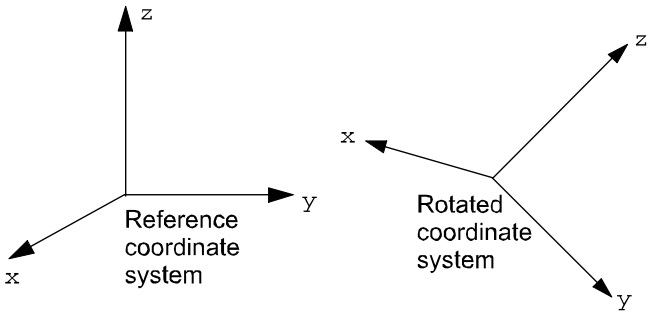
$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$$

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Co je kvaternion?

Orientace souřadného systému (např. nástroje) je popisována rotační maticí, která popisuje směr os souřadného systému ve vztahu k referenčnímu systému (viz následující obrázek).



xx0500002376

Otočené osy souřadného systému (x, y, z) jsou vektory, které mohou být vyjádřeny v referenčním souřadném systému takto:

x = (x1, x2, x3)

y = (y1, y2, y3)

z = (z1, z2, z3)

To znamená, že x-komponent x-vektoru v referenčním souřadném systému bude x1, y-komponent bude x2 a tak dále.

Tyto tři vektory mohou být sloučeny dohromady v matici (rotační matice), kde každý z vektorů tvoří jeden ze sloupců:

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}$$

xx0500002381

Kvaternion je jen výstižnější způsob, jak popsat tuto rotační matici; kvaterniony se vypočítávají na základě elementů rotační matice:

q1 = $\frac{\sqrt{x_1+y_2+z_3+1}}{2}$	
q2 = $\frac{\sqrt{x_1-y_2-z_3+1}}{2}$	sign q2 = sign (y3-z2)
q3 = $\frac{\sqrt{y_2-x_1-z_3+1}}{2}$	sign q3 = sign (z1-x3)
q4 = $\frac{\sqrt{z_3-x_1-y_2+1}}{2}$	sign q4 = sign (x2-y1)

Pokračování na další straně

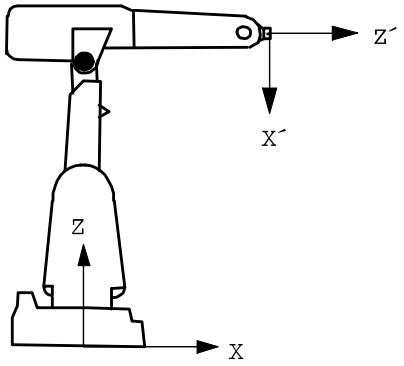
© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.52 orient - Orientace
RobotWare - OS
Pokračování

Příklad 1

Nástroj je orientován tak, že jeho osa Z' směřuje přímo dopředu (ve stejném směru jako osa X souřadného systému základny). Osa Y' nástroje odpovídá ose Y souřadného systému základny (viz následující obrázek). Jak je orientace nástroje definována v pozičních datech (`robtarget`)?

Orientace nástroje v naprogramované pozici je normálně vztažena k souřadnému systému použitého pracovního objektu. V tomto příkladu není použit žádný pracovní objekt a souřadný systém základny je totožný se světovým souřadným systémem. Tudíž, orientace je vztažena k souřadnému systému základny.



xx0500002377

Osy budou potom souviset takto:

x' = -z = (0, 0, -1)

y' = y = (0, 1, 0)

z' = x = (1, 0, 0)

Což odpovídá následující rotační matici:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

xx0500002388

Rotační matice poskytuje odpovídající kvaternion:

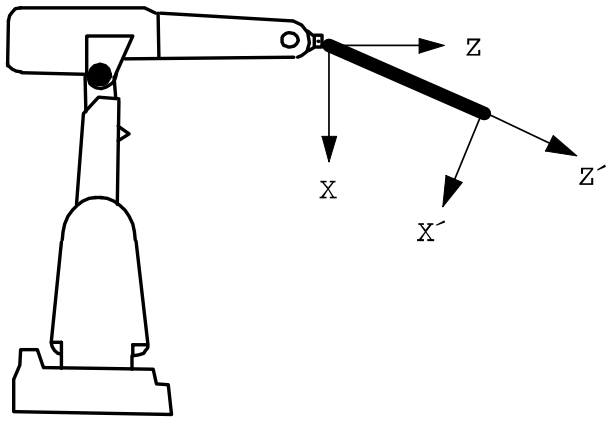
q1 = $\frac{\sqrt{0+1+0+1}}{2}$ = $\frac{\sqrt{2}}{2}$ = 0.707	
q2 = $\frac{\sqrt{0-1-0+1}}{2}$ = 0	
q3 = $\frac{\sqrt{1-0-0+1}}{2}$ = $\frac{\sqrt{2}}{2}$ = 0.707	sign q3 = sign (1+1) = +
q4 = $\frac{\sqrt{0-0-1+1}}{2}$ = 0	

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Příklad 2

Směr nástroje je otočen o 30° ohledně os X' a Z' ve vztahu k souřadnému systému zápěstí (viz následující obrázek). Jak je orientace nástroje definována v datech nástroje?



xx0500002378

Osy budou potom souviset takto:

x' = (cos30°, 0, -sin30°)

y' = (0, 1, 0)

z' = (sin30°, 0, cos30°)

Což odpovídá následující rotační matici:

$$\begin{bmatrix} \cos 30° & 0 & \sin 30° \\ 0 & 1 & 0 \\ -\sin 30° & 0 & \cos 30° \end{bmatrix}$$

xx0500002393

Rotační matice poskytuje odpovídající kvaternion:

q1 = $\frac{\sqrt{cos30°+1+cos30°+1}}{2}$ = 0.965926	
q2 = $\frac{\sqrt{cos30°-1-cos30°+1}}{2}$ = 0	
q3 = $\frac{\sqrt{1-cos30°-cos30°+1}}{2}$ = 0.258819	sign q3 = sign (sin30°+sin30°) = +
q4 = $\frac{\sqrt{cos30°-cos30°-1+1}}{2}$ = 0	

Konstrukce

```
< dataobject of orient >
  < q1 of num >
  < q2 of num >
  < q3 of num >
  < q4 of num >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.52 orient - Orientace
RobotWare - OS
Pokračování

Související informace

Pro informace o	Viz
Operace na orientacích	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Výrazy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.53 paridnum - Typ identifikace parametru

Použití

`paridnum` se používá k zastoupení celého čísla symbolickou konstantou.

Popis

Konstanta `paridnum` je určena k používání pro identifikaci parametru, jako je zátěžová identifikace nástroje nebo užitečné zátěže nebo zátěže externího manipulátoru.

Základní příklady

Následující příklad názorně ukazuje datový typ `paridnum`:

Příklad 1

```
TEST ParIdRobValid (TOOL_LOAD_ID)
  CASE ROB_LOAD_VAL:
  ! Possible to do load identification of tool in actual robot type
  ...
  CASE ROB_LM1_LOAD_VAL:
  ! Only possible to do load identification of tool with
  ! IRB 6400FHD if actual load < 200 kg
  ...
  CASE ROB_NOT_LOAD_VAL:
  ! Not possible to do load identification of tool in actual robot
       type
  ...
ENDTEST
```

Použití předdefinované konstanty `TOOL_LOAD_ID` datového typu `paridnum`.

Předdefinovaná data

Následující symbolické konstanty datového typu `paridnum` jsou předdefinovány a používají se jako argumenty v následujících instrukcích `ParIdRobValid`, `ParIdPosValid`, `LoadId` a `ManLoadIdProc`.

Hodnota	Symbolická konstanta	Komentář
1	TOOL_LOAD_ID	Identifikovat zatížení nástroje
2	PAY_LOAD_ID	Identifikovat užitečnou zátěž (Ref. instrukce GripLoad)
3	IRBP_K	Identifikovat externí manipulátor IRBP K zátěž
4	IRBP_L	Identifikovat externí manipulátor IRBP L zátěž
4	IRBP_C	Identifikovat externí manipulátor IRBP C zátěž
4	IRBP_C_INDEX	Identifikovat externí manipulátor IRBP C_INDEX zátěž
4	IRBP_T	Identifikovat externí manipulátor IRBP T zátěž
5	IRBP_R	Identifikovat externí manipulátor IRBP R zátěž
6	IRBP_A	Identifikovat externí manipulátor IRBP A zátěž
6	IRBP_B	Identifikovat externí manipulátor IRBP B zátěž
6	IRBP_D	Identifikovat externí manipulátor IRBP D zátěž

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.53 paridnum - Typ identifikace parametru

RobotWare - OS

Pokračování

> **POZNÁMKA**
>
> Pouze `TOOL_LOAD_ID` a `PAY_LOAD_ID` se používá v uživatelsky definovaných programech RAPID k identifikaci zátěže nástroje resp. užitečné zátěže u robotu.

Charakteristika

`paridnum` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Předddefinovaný program Load Identify	*Návod k použití - IRC5 s jednotkou FlexPendant*, sekce *Programování a testování - Servisní rutiny - LoadIdentify, identifikace zátěže a servisní rutiny*
Platný typ robotu	*ParIdRobValid - Platný typ robotu pro identifikaci parametru na str 1250*
Platná pozice robotu	*ParIdPosValid - Platná pozice robotu pro identifikaci parametru na str 1247*
Identifikace zátěže s kompletním příkladem	*LoadId - Identifikace zatížení nástroje nebo užitečné zátěže na str 332*
Identifikace zátěže externích manipulátorů	*ManLoadIdProc - Identifikace zátěže IRBP manipulátorů na str 339*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.54 paridvalidnum - Výsledek ParIdRobValid

Použití

`paridvalidnum` se používá k zastoupení celého čísla symbolickou konstantou.

Popis

Konstanta `paridvalidnum` je určena k používání pro identifikaci parametru, jako je zátěžová identifikace nástroje nebo užitečné zátěže při kontrole vrácené hodnoty od funkce `ParIdRobValid`.

Základní příklady

Následující příklady názorně ukazují datový typ `paridvalidnum`:

```
TEST ParIdRobValid (PAY_LOAD_ID)
  CASE ROB_LOAD_VAL:
  ! Possible to do load identification of payload in actual robot
  ! type
  ...
  CASE ROB_LM1_LOAD_VAL:
  ! Only possible to do load identification of payload
  ! with IRB 6400FHD if actual load < 200 kg
  ...
  CASE ROB_NOT_LOAD_VAL:
  ! Not possible to do load identification of payload
  ! in actual robot type
  ...
ENDTEST
```

Použití předdefinovaných konstant `ROB_LOAD_VAL`, `ROB_LM1_LOAD_VAL` a `ROB_NOT_LOAD_VAL` datového typu `paridvalidnum`.

Předdefinovaná data

Následující symbolické konstanty datového typu `paridvalidnum` jsou předdefinovány a používají se pro kontrolu vrácené hodnoty od funkce `ParIdRobValid`

Hodnota	Symbolická konstanta	Komentář
10	ROB_LOAD_VAL	Platný typ robotu pro identifikaci aktuálního parametru.
11	ROB_NOT_LOAD_VAL	Neplatný typ robotu pro identifikaci aktuálního parametru.
12	ROB_LM1_LOAD_VAL	Platný typ robotu IRB 6400FHD pro identifikaci aktuálního parametru, jestliže aktuální zátěž < 200 kg.

Charakteristika

`paridvalidnum` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.54 paridvalidnum - Výsledek ParIdRobValid

RobotWare - OS

Pokračování

Související informace

Pro informace o	Viz
Předdefinovaný program Load Identify	*Návod k použití - IRC5 s jednotkou FlexPendant*, sekce *Programování a testování - Servisní rutiny - LoadIdentify, identifikace zátěže a servisní rutiny*
Platný typ robotu	*ParIdRobValid - Platný typ robotu pro identifikaci parametru na str 1250*
Platná pozice robotu	*ParIdPosValid - Platná pozice robotu pro identifikaci parametru na str 1247*
Identifikace zátěže s kompletním příkladem	*LoadId - Identifikace zatížení nástroje nebo užitečné zátěže na str 332*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.55 pathrecid - Identifikátor záznamníku dráhy

Použití

`pathrecid` se používá k identifikaci bodu přerušení pro záznamník dráhy.

Popis

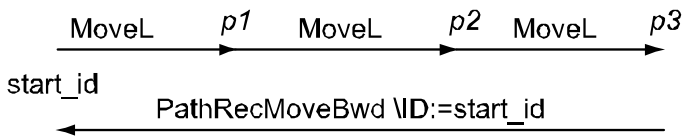
Záznamník dráhy je systémová funkce pro záznam vykonávané dráhy robotu. Data typu `pathrecid` mohou být propojena s konkrétním místem dráhy prostřednictvím instrukce `PathRecStart`. Uživatel může potom přikázat záznamník k provedení pohybu zpět k identifikátoru dráhy pomocí instrukce `PathRecMoveBwd`.

Základní příklady

Následující příklad názorně ukazuje datový typ `pathrecid`:

Příklad 1

```
VAR pathrecid start_id;
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];

PathRecStart start_id;
MoveL p1, vmax, z50, tool1;
MoveL p2, vmax, z50, tool1
MoveL p3, vmax, z50, tool1;
IF(PathRecValidBwd (\ID := start_id)) THEN
  StorePath;
  PathRecMoveBwd \ID:=start_id;
  ...
ENDIF
```



pathrecid_Ex

Předcházející příklad spustí záznamník dráhy a bod startu bude označen identifikátorem dráhy `start_id`. Potom se robot bude pohybovat dopředu s tradičními pohybovými instrukcemi a potom se bude pohybovat znovu zpět ke startovní pozici s pomocí zaznamenané dráhy. Aby bylo možné provést pohybové instrukce PathRecorder, úroveň dráhy je nutné změnit s `StorePath`.

Charakteristika

`pathrecid` je nehodnotový datový typ.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.55 pathrecid - Identifikátor záznamníku dráhy

Path Recovery

Pokračování

Související informace

Pro informace o	Viz
Spustit - zastavit záznamník dráhy	*PathRecStart - Spustit záznamník dráhy na str 467* *PathRecStop - Zastavit záznamník dráhy na str 470*
Zkontrolovat platnou zaznamenanou dráhu	*PathRecValidBwd - Je zaznamenána platná zpětná dráha na str 1255* *PathRecValidFwd - Je zaznamenána platná dráha dopředu na str 1259*
Přehrát záznamník dráhy dozadu	*PathRecMoveBwd - Posunout záznamník dráhy dozadu na str 458*
Přehrát záznamník dráhy dopředu	*PathRecMoveFwd - Posunout záznamník dráhy dopředu na str 464*
Vlastnosti nehodnotových datových typů	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.56 pos - Pozice (pouze X, Y a Z)

Použití

`pos` se používá pro pozice (pouze X, Y a Z).

Datový typ `robtarget` se používá pro pozici robotu včetně orientace nástroje a konfigurace os.

Popis

Data typu `pos` popisují souřadnice pozice: X, Y a Z.

Komponenty

Datový typ `pos` má následující komponenty:

`x`

Datový typ: `num`

X-hodnota pozice.

`y`

Datový typ: `num`

Y-hodnota pozice.

`z`

Datový typ: `num`

Z-hodnota pozice.

Základní příklady

Následující příklady názorně ukazují datový typ `pos`:

Příklad 1

```
VAR pos pos1;
...
pos1 := [500, 0, 940];
```

Pozici `pos1` je přidělena hodnota: X=500 mm, Y=0 mm, Z=940 mm.

Příklad 2

```
pos1.x := pos1.x + 50;
```

Pozice `pos1` je posunuta o `50` mm ve směru X.

Konstrukce

```
< dataobject of pos >
  < x of num >
  < y of num >
  < z of num >
```

Související informace

Pro informace o	Viz
Operace na pozicích	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Výrazy*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.56 pos - Pozice (pouze X, Y a Z)

RobotWare - OS

Pokračování

Pro informace o	Viz
Pozice robotu včetně orientace	*robtarget - Poziční data na str 1572*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.57 pose - Transformace souřadnic

Použití

`pose` se používá ke změně z jednoho souřadného systému na jiný.

Popis

Data typu `pose` popisují, jak je souřadný systém posunut a otočen kolem jiného souřadného systému. Data mohou, například, popisovat, jak je umístěn a orientován souřadný systém nástroje ve vztahu k souřadnému systému zápěstí.

Komponenty

Datový typ má následující komponenty:

`trans`

translation

Datový typ: `pos`

Posun v pozici (x, y a z) souřadného systému.

`rot`

rotation

Datový typ: `orient`

Otáčení souřadného systému.

Základní příklady

Následující příklady názorně ukazují datový typ `pose`:

```
VAR pose frame1;
...
frame1.trans := [50, 0, 40];
frame1.rot := [1, 0, 0, 0];
```

Transformaci souřadnic `frame1` je přidělena hodnota, která odpovídá posunu v pozici, kde X= `50` mm, Y=`0` mm, Z=`40` mm; tam je, nicméně, žádná rotace.

Konstrukce

```
< dataobject of pose >
  < trans of pos >
  < rot of orient >
```

Související informace

Pro informace o	Viz
Co je kvaternion?	*orient - Orientace na str 1542*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.58 processtimes - procesní časy

Použití

`processtimes` se používá k definování dob trvání u všech fází dohledu stavu v CAP, kromě fáze MAIN, která je definována pohybem robotu (viz sekce *Dohled* v *Application manual - Continuous Application Platform*).

`processtimes` je komponent `capdata` a definuje časové limity pro následující fáze dohledu stavu v CAP:

- PRE_START
- POST1
- POST2

Určený časový limit musí být delší než nula, jestliže dohled by měl být použit během odpovídající fáze dohledu stavu v CAP (viz sekce *Dohled a procesní fáze* v *Application manual - Continuous Application Platform*).

Komponenty

pre

Datový typ: `num`

Definuje trvání fáze PRE_START v sekundách. Během té doby musí být splněny všechny podmínky definované pro tuto fázi.

post1

Datový typ: `num`

Definuje trvání fáze POST1 v sekundách. Během té doby musí být splněny všechny podmínky definované pro tuto fázi.

post2

Datový typ: `num`

Definuje trvání fáze POST2 v sekundách. Během té doby musí být splněny všechny podmínky definované pro tuto fázi.

Syntaxe

```
< data object of processtimes >
  < pre of num >
  < post1 of num >
  < post2 of num >
```

Související informace

	Popsáno v:
Datový typ `capdata`	*capdata - CAP data na str 1450*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.59 progdisp - Posun programu

Použití

`progdisp` se používá k uložení aktuálního posunu programu robotu a externích os.

Tento datový typ se nemusí normálně používat, jelikož data jsou nastavena pomocí instrukcí `PDispSet`, `PDispOn`, `PDispOff`, `EOffsSet`, `EOffsOn` a `EOffsOff`. Používá se pouze k dočasnému uložení aktuální hodnoty pro pozdější použití.

Popis

K aktuálním hodnotám posunu programu se přistupuje pomocí systémové proměnné `C_PROGDISP`.

Další informace najdete v instrukcích `PDispSet`, `PDispOn`, `EOffsSet` a `EOffsOn`.

Komponenty

`pdisp`

program displacement

Datový typ: `pose`

Posun programu u robotu vyjádřený pomocí překladu a orientace. Překlad je vyjádřen v mm.

`eoffs`

external offset

Datový typ: `extjoint`

Ofset pro každou z externích os. Jestliže osa je lineární, hodnota je vyjádřena v mm; jestliže je rotační, hodnota je vyjádřena ve stupních.

Základní příklady

Následující příklad názorně ukazuje datový typ `progdisp`:

Příklad 1

```
VAR progdisp progdisp1;
...
SearchL sen1, psearch, p10, v100, tool1;
PDispOn \ExeP:=psearch, *, tool1;
EOffsOn \ExeP:=psearch, *;
...
progdisp1:=C_PROGDISP;
PDispOff;
EOffsOff;
...
PDispSet progdisp1.pdisp;
EOffsSet progdisp1.eoffs;
```

Nejprve je posun programu aktivován od hledané pozice. Potom jsou aktuální hodnoty posunu programu dočasně uloženy do proměnné `progdisp1` a posun

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

programu je deaktivován. Později je provedena nová aktivace pomocí instrukcí `PDispSet` a `EOffsSet`.

Předdefinovaná data

Systémová proměnná `C_PROGDISP` popisuje aktuální posun programu robotu a externích os a může se k ní vždy přistupovat z programu. Na druhé straně je možné ji měnit pouze pomocí řady instrukcí, nikoliv přidělením.

Následující výchozí hodnoty pro posun programu jsou nastaveny

- při používání restartovacího režimu **Reset RAPID**
- při načítání nového programu nebo nového modulu
- při spuštění vykonávání programu od začátku
- při posunu ukazatele programu k `main`
- při posunu ukazatele programu k rutině
- při posunu ukazatele programu takovým způsobem, že pořadí provádění je ztraceno.

```
VAR progdisp C_PROGDISP :=
  [ [[ 0, 0, 0], [1, 0, 0, 0]],-> posedata
  [ 0, 0, 0, 0, 0, 0]];-> extjointdata
```

Konstrukce

```
< dataobject of progdisp >
  < pdisp of pose >
    < trans of pos >
      < x of num >
      < y of num >
      < z of num >
    < rot of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
  < eoffs of extjoint >
    < eax_a of num >
    < eax_b of num >
    < eax_c of num >
    < eax_d of num >
    < eax_e of num >
    < eax_f of num >
```

Související informace

Pro informace o	Viz
Instrukce pro definování posunu programu	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Nastavení pohybu*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Souřadné systémy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.60 rawbytes - Data raw

Použití

`rawbytes` se používá jako obecný datový kontejner. Může se používat pro komunikaci s I/O zařízeními.

Popis

Data `rawbytes` mohou být vyplněna každým typem dat - `num`, `byte`, `string` - prostřednictvím podpůrných instrukcí/funkcí. Do každé proměnné `rawbytes` systém také ukládá aktuální délku platných bajtů.

Základní příklady

Následující příklad názorně ukazuje datový typ `rawbytes`:

Příklad 1

```
VAR rawbytes raw_data;
VAR num integer := 8;
VAR num float := 13.4;

ClearRawBytes raw_data;
PackRawBytes integer, raw_data, 1 \IntX := INT;
PackRawBytes float, raw_data, (RawBytesLen(raw_data)+1) \Float4;
```

V tomto příkladu proměnná `raw_data` typu `rawbytes` je nejprve vyčištěna, tj. všechny bajty jsou nastaveny na 0 (stejné jako výchozí nastavení při deklaraci). Potom je do prvních 2 bajtů umístěna hodnota `integer` a do dalších 4 bajtů hodnota `float`.

Omezení

Proměnná `rawbytes` může obsahovat 0 až 1024 bajty.

Konstrukce

`rawbytes` je nehodnotový datový typ.

Při deklaraci proměnné `rawbytes`jsou všechny bajty v `rawbytes` nastaveny na 0 a aktuální délka platných bajtů je nastavena na 0.

Související informace

Pro informace o	Viz
Získat délku dat `rawbytes`	*RawBytesLen - Získat délku dat rawbytes na str 1278*
Vyčistit obsah dat `rawbytes`	*ClearRawBytes - Vyčistit obsahy rawbyte dat na str 118*
Kopírovat obsah dat `rawbytes`	*CopyRawBytes - Kopírovat obsah dat rawbytes na str 137*
Zabalit hlavičku DeviceNet do dat `rawbytes`	*PackDNHeader - Zabalit hlavičku DeviceNet do dat rawbytes na str 446*
Zabalit data do dat `rawbytes`	*PackRawBytes - Zabalit data do dat rawbytes na str 449*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.60 rawbytes - Data raw

RobotWare - OS

Pokračování

Pro informace o	Viz
Zapsat data `rawbytes`	*WriteRawBytes - Zapsat data rawbytes na str 994*
Načíst data `rawbytes`	*ReadRawBytes - Přečíst data rawbytes na str 530*
Rozbalit data z dat `rawbytes`	*UnpackRawBytes - Rozbalit data z rawbyte dat na str 908*
Manipulace se souborem a sériovým kanálem	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.61 restartblkdata - bloková data pro restart

Použití

`restartblkdata` se používá k definování chování CAP procesu při restartu.

`restartblkdata` je komponent `capdata` a definuje následující pro CAP proces při restartu, jestliže:

- Robot by měl provést/blokovat stacionární weaving během procesu restartu (`weave_start`).
- Restart pohybu robotu by měl být zpožděn nebo restart nesouvisejícího procesu (`motion_delay`).
- Fáze PRE a fáze PRE_START by měly být provedeny/blokovány (`pre_phase`).
- Rychlost odlišná od hlavní rychlosti by měla být použita nebo nikoliv během startu procesu (`startspeed_phase`).
- Fáze START POST1 a fáze POST1 by měly být provedeny/blokovány (`post1_phase`).
- Fáze START POST2 a fáze POST2 by měly být provedeny/blokovány (`post2_phase`).

Komponenty

weave_start

Datový typ: `bool`

Hodnota	Popis
FALSE	Stacionární weaving při restartu až do startu procesu
TRUE	Žádný stacionární weaving při restartu až do startu procesu

motion_delay

Datový typ: `bool`

Hodnota	Popis
FALSE	Prodleva pohybu robotu při restartu po startu procesu
TRUE	Žádná prodleva pohybu robotu při restartu po startu procesu

pre_phase

Datový typ: `bool`

Hodnota	Popis
FALSE	Proveďte fázi PRE a fázi PRE_START při restartu
TRUE	NEPROVÁDĚJTE fázi PRE a fázi PRE_START při restartu

startspeed_phase

Datový typ: `bool`

Hodnota	Popis
FALSE	Posuňte robot počáteční rychlostí na začátku restartu
TRUE	NEPOSUNUJTE robot počáteční rychlostí na začátku restartu, použijte přímo hlavní rychlost

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.61 restartblkdata - bloková data pro restart

Continuous Application Platform (CAP)
Pokračování

post1_phase

Datový typ: `bool`

Hodnota	Popis
FALSE	Proveďte fázi START_POST1 a fázi POST1 při restartu
TRUE	NEPROVÁDĚJTE fázi START_POST1 a fázi POST1 při restartu

post2_phase

Datový typ: `bool`

Hodnota	Popis
FALSE	Proveďte fázi START_POST2 a fázi POST2 při restartu
TRUE	NEPROVÁDĚJTE fázi START_POST2 a fázi POST2 při restartu

Syntaxe

```
< data object of restartblkdata >
  < weave_start of bool >
  < motion_delay of bool >
  < pre_phase of bool >
  < startspeed_phase of bool >
  < post1_phase of bool >
  < post2_phase of bool >
```

Související informace

	Popsáno v:
Datový typ `capdata`	*capdata - CAP data na str 1450*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.62 restartdata - Data restartu pro signály trigg

Použití

`restartdata` zrcadlí před- a po-hodnoty určených I/O signálů (procesní signály) při stop sekvenci pohybů robotu. I/O signály k dohledu jsou určeny v instrukci `TriggStopProc`.

`TriggStopProc` a `restartdata` jsou určeny pro použití k restartu po zastavení programu (STOP) nebo nouzovém zastavení (QSTOP) vlastních procesních instrukcí definovaných v RAPIDu (rutiny NOSTEPIN).

Definice

Tabulka ukazuje definici časového bodu pro čtení před- a po-hodnot pro I/O signály.

Typ zastavení	Přečíst čas pro předhodnotu I/O signálu	Přečíst čas pro pohodnotu I/O signálu
STOP na dráze	Když všechny osy robotu stojí v klidu	Asi 400 ms po před-času
QSTOP mimo dráhu	Co nejdříve	Asi 400 ms po před-času

Popis

`restartdata` zrcadlí následující data po zastavení vykonávání programu:

- platná data restartu
- robot se zastavil na dráze nebo nikoliv
- předhodnota I/O signálů
- pohodnota I/O signálů
- počet boků mezi před-časem a post-časem stínového signálu pro probíhající proces

Komponenty

`restartstop`

valid restartdata after stop

Datový typ: `bool`

TRUE = Zrcadlit poslední STOP nebo QSTOP

FALSE = Neplatná data restartu. Všechny hodnoty I/O signálů jsou nastaveny na -1.

`stoponpath`

stop on path

Datový typ: `bool`

TRUE = Robot je zastaven na dráze (STOP)

FALSE = Robot je zastaven, ale nikoliv na dráze (QSTOP)

`predo1val`

pre do1 value

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.62 restartdata - Data restartu pro signály trigg
RobotWare - OS
Pokračování

Datový typ: `dionum`

Předhodnota digitálního signálu “do1” určená v argumentu `DO1` v instrukci `TriggStopProc`.

`postdo1val`

post do1 value

Datový typ: `dionum`

Posthodnota digitálního signálu “do1” určená v argumentu `DO1` v instrukci `TriggStopProc`.

`prego1val`

pre go1 value

Datový typ: `num`

Předhodnota digitálního skupinového signálu “go1” určená v argumentu `GO1` v instrukci `TriggStopProc`.

`postgo1val`

post go1 value

Datový typ: `num`

Posthodnota digitálního skupinového signálu “go1” určená v argumentu `GO1` v instrukci `TriggStopProc`.

`prego2val`

pre go2 value

Datový typ: `num`

Předhodnota digitálního skupinového signálu “go2” určená v argumentu `GO2` v instrukci `TriggStopProc`.

`postgo2val`

post go2 value

Datový typ: `num`

Posthodnota digitálního skupinového signálu “go2” určená v argumentu `GO2` v instrukci `TriggStopProc`.

`prego3val`

pre go3 value

Datový typ: `num`

Předhodnota digitálního skupinového signálu “go3” určená v argumentu `GO3` v instrukci `TriggStopProc`.

`postgo3val`

post go3 value

Datový typ: `num`

Posthodnota digitálního skupinového signálu “go3” určená v argumentu `GO3` v instrukci `TriggStopProc`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`prego4val`

pre go4 value

Datový typ: `num`

Předhodnota digitálního skupinového signálu “go4” určená v argumentu `GO4` **v instrukci** `TriggStopProc`.

`postgo4val`

post go4 value

Datový typ: `num`

Posthodnota digitálního skupinového signálu “go4” určená v argumentu `GO4` **v instrukci** `TriggStopProc`.

`preshadowval`

pre shadow value

Datový typ: `dionum`

Předhodnota „stínu“ digitálního signálu určená v argumentu `ShadowDO` **v instrukci** `TriggStopProc`.

`shadowflanks`

number of shadow flanks

Datový typ: `num`

Počet hodnotových přechodů (boků) „stínu“ digitálního signálu mezi předčasem a postčasem. „Stín“ signálu je určen v argumentu `ShadowDO` **v instrukci** `TriggStopProc`.

`postshadowval`

post shadow value

Datový typ: `dionum`

Posthodnota „stínu“ digitálního signálu určená v argumentu `ShadowDO` **v instrukci** `TriggStopProc`.

Konstrukce

```
< dataobject of restartdata >
  < restartstop of bool >
  < stoponpath of bool >
  < predo1val of dionum >
  < postdo1val of dionum >
  < prego1val of num >
  < postgo1val of num >
  < prego2val of num >
  < postgo2val of num >
  < prego3val of num >
  < postgo3val of num >
  < prego4val of num >
  < postgo4val of num >
  < preshadowval of dionum >
  < shadowflanks of dionum >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.62 restartdata - Data restartu pro signály trigg

RobotWare - OS

Pokračování

```
< postshadowval of dionum >
```

Související informace

Pro informace o	Viz
Instrukce k předdefinovanému procesu	*TriggL - Lineární pohyby robotu s událostmi na str 839* *TriggC - Kruhový pohyb robotu s událostmi na str 796*
Nastavení zrcadla dat restartu	*TriggStopProc - Generovat restartovací data pro trigg signály při zastavení na str 876*
Provést pohyb zpět na dráze	*StepBwdPath - Posunout zpět o jeden krok na dráze na str 720*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.63 rmqheader - Hlavička fronty zpráv RAPID

Použití

`rmqheader` (*RAPID Message Queue Header*) se používá pro čtení datové struktury ve zprávě typu `rmqmessage`.

Popis

Hlavičková část nehodnotového datového typu `rmqmessage` převedeného na datový typ `rmqheader`.

Komponenty

`datatype`

Datový typ: `string`

Jméno použitého datového typu, např. `num`, `string` nebo jiného hodnotového datového typu.

`ndim`

Number of Dimensions

Datový typ: `num`

Počet rozměrů pole.

`dim1`

Size of first dimension

Datový typ: `num`

Velikost prvního rozměru. 0, pokud se nepoužívá.

`dim2`

Size of second dimension

Datový typ: `num`

Velikost druhého rozměru. 0, pokud se nepoužívá.

`dim3`

Size of third dimension

Datový typ: `num`

Velikost třetího rozměru. 0, pokud se nepoužívá.

Příklady

Základní příklady datového typu `rmqheader` jsou názorně uvedeny dole.

Příklad 1

```
VAR rmqmessage message;
VAR rmqheader header;
...
RMQGetMessage message;
RMQGetMsgHeader message \Header:=header;
```

Kopírovat a převést informaci `rmqheader` ze zprávy `rmqmessage`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.63 rmqheader - Hlavička fronty zpráv RAPID

FlexPendant Interface, PC Interface, or Multitasking

Pokračování

Konstrukce

```
<dataobject of rmqheader>
  <datatype of string>
  <ndim of num>
  <dim1 of num>
  <dim2 of num>
  <dim3 of num>
```

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
Vyjímá hlavičku dat z `rmqmessage`	*RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ na str 562*
`RMQ Message`	*rmqmessage - Zpráva z fronty zpráv RAPID na str 1569*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.64 rmqmessage - Zpráva z fronty zpráv RAPID

Použití

`rmqmessage` (*RAPID Message Queue Message*) se používá k dočasnému uložení komunikačních dat.

Popis

Datový typ `rmqmessage` je zpráva používaná k uložení dat při komunikaci mezi různými úlohami RAPID nebo klienty Robot Application Builder s funkčností RMQ. Obsahuje informace o typu odeslaných dat, rozměrech dat, identitě odesílatele a konkrétní data.

`rmqmessage` je velký datový typ (asi 3000 bajtů) a doporučuje se, aby proměnná byla znovu použita pro uložení paměti RAPID.

Základní příklady

Následující příklad názorně ukazuje datový typ `rmqmessage`:

Příklad 1

```
VAR rmqmessage rmqmessage1;
VAR string myrecdata;
...
RMQGetMsgData rmqmessage1, myrecdata;
```

Proměnná `rmqmessage1` je definována a může se použít v příkazu RMQ (fronta zpráv RAPID). V tomto příkladu je datová část v `rmqmessage1` zkopírována do proměnné `myrecdata`.

Charakteristika

`rmqmessage` je nehodnotový datový typ a nemůže se používat v operacích orientovaných na hodnotu.

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
`RMQ Header`	*rmqheader - Hlavička fronty zpráv RAPID na str 1567*
Vyjímá hlavičku dat z `rmqmessage`	*RMQGetMsgHeader - Získat informace z hlavičky zprávy RMQ na str 562*
Přikázat a zapnout přerušení pro určený datový typ	*IRMQMessage - Přikazuje přerušení RMQ pro datový typ na str 288*
Získat první zprávu z fronty zpráv RAPID (RMQ).	*RMQGetMessage - Získat zprávu RMQ na str 556*
Odeslat data do fronty úlohy RAPID nebo klienta Robot Application Builder a čekat na odpověď od klienta.	*RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu na str 572*
Vyjímá data z `rmqmessage`	*RMQGetMsgData - Získat datovou část zprávy RMQ na str 559*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.65 rmqslot - Číslo identity klienta RMQ

Použití

`rmqslot` (*RAPID Message Queue Slot*) se používá při komunikaci s RMQ nebo klientem Robot Application Builder.

Popis

`rmqslot` je číslo identity fronty zpráv RAPID konfigurované pro úlohu RAPID nebo číslo identity klienta Robot Application Builder.

Základní příklady

Následující příklad názorně ukazuje datový typ `rmqslot`:

Příklad 1

```
VAR rmqslot rmqslot1;
RMQFindSlot rmqslot1, "RMQ_T_ROB1";
...
```

Proměnná `rmqslot1` je definována a může se použít v instrukci `RMQFindSlot` k získání čísla identity fronty zpráv RAPID "`RMQ_T_ROB1`" konfigurované pro úlohu RAPID "`T_ROB1`".

Charakteristika

`rmqslot` je nehodnotový datový typ a nemůže se používat v operacích orientovaných na hodnotu.

Související informace

Pro informace o	Viz
Popis funkce RAPID o frontě zpráv (RMQ)	*Application manual - Controller software IRC5*, sekce *Fronta zpráv RAPID*.
Najděte číslo identity úlohy fronty zpráv RAPID nebo klienta Robot Application Builder.	*RMQFindSlot - Najít identitu slotu od jména slotu na str 554*
Odeslat data do fronty úlohy RAPID nebo klienta Robot Application Builder	*RMQSendMessage - Odeslat datovou zprávu RMQ na str 568*
Odešlete data klientu a čekejte na odpověď od klienta.	*RMQSendWait - Odeslat datovou zprávu RMQ a čekat na odezvu na str 572*
Získat jméno slotu od určené identity slotu	*RMQGetSlotName - Získat jméno klienta RMQ na str 1302*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.66 robjoint - Společná pozice os robotu

Použití

`robjoint` se používá pro definování pozice os robotu ve stupních.

Popis

Data typu `robjoint` se používají k uložení pozic os robotu 1 až 6 ve stupních. Pozice osy je definována jako rotace ve stupních pro příslušnou osu (rameno) v kladném nebo záporném směru od kalibrační pozice osy.

Komponenty

`rax_1`

robot axis 1

Datový typ: `num`

Pozice osy 1 robotu ve stupních od kalibrační pozice.

`...`

`rax_6`

robot axis 6

Datový typ: `num`

Pozice osy 6 robotu ve stupních od kalibrační pozice.

Konstrukce

```
< dataobject of robjoint >
  < rax_1 of num >
  < rax_2 of num >
  < rax_3 of num >
  < rax_4 of num >
  < rax_5 of num >
  < rax_6 of num >
```

Související informace

Pro informace o	Viz
Data společné pozice	*jointtarget - Data pozice svaru na str 1520*
Přesunout do pozice svaru	*MoveAbsJ - Posune robot do absolutní pozice spoje na str 352*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.67 robtarget - Poziční data

Použití

`robtarget` (*robot target*) se používá pro definování pozice robotu a pomocných os.

Popis

Poziční data se používají k definování pozice v pohybových instrukcích, ke které se robot a pomocné osy budou pohybovat.

Jelikož robot je schopen dosáhnout stejné pozice několika různými cestami, je určena také konfigurace osy. Toto definuje hodnoty osy, jestliže jsou nějakým způsobem nejednoznačné, například:

- jestliže robot je v pozici dopředu nebo dozadu,
- jestliže osa 4 směřuje dolů nebo nahoru,
- jestliže osa 6 má zápornou nebo kladnou otáčku.

VAROVÁNÍ

Pozice je definována na základě souřadného systému pracovního objektu včetně jakéhokoliv posunu programu. Jestliže pozice je naprogramována s nějakým jiným pracovním objektem, než který byl použit instrukci, robot se nebude pohybovat očekávaným způsobem. Ujistěte se, že používáte stejný pracovní objekt, který byl použit při programování pohybových instrukcí. Nesprávné použití může někoho zranit nebo poškodit robot nebo jiné zařízení.

Komponenty

`trans`

translation

Datový typ: `pos`

Pozice (x, y a z) středního bodu nástroje vyjádřená v mm.

Pozice je určena v souvislosti se souřadným systémem aktuálního objektu včetně posunu programu. Jestliže není určen žádný pracovní objekt, potom je to světový souřadný systém.

`rot`

rotation

Datový typ: `orient`

Orientace nástroje vyjádřená formou kvaternionu (q1, q2, q3 a q4).

Orientace je určena v souvislosti se souřadným systémem aktuálního objektu včetně posunu programu. Jestliže není určen žádný pracovní objekt, potom je to světový souřadný systém.

`robconf`

robot configuration

Datový typ: `confdata`

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Konfigurace osy robotu (`cf1`, `cf4`, `cf6` a `cfx`). Toto je definováno formou aktuální čtvrtotáčky osy 1, osy 4 a osy 6. První kladná čtvrtotáčka 0 až 90° je definována jako 0. Význam komponentu `cfx` závisí na typu robotu.

Pro více informací viz datový typ `confdata`.

`extax`

external axes

Datový typ: `extjoint`

Pozice pomocných os.

Pozice `is defined` následně pro každou jednotlivou osu (`eax_a`, `eax_b...eax_f`):

- Rotační osy - pozice je definována jako otočení ve stupních od kalibrační pozice.
- Lineární osy - pozice je definována jako vzdálenost v mm od kalibrační pozice.

Pomocné osy `eax_a ...` jsou logickými osami. Jaký vztah mezi sebou má číslo logické osy a číslo fyzické osy, to je definováno v systémových parametrech.

Hodnota 9E9 je definována pro osy, které nejsou připojeny. Jestliže se liší osy definované v pozičních datech od os, které jsou aktuálně připojeny při vykonávání programu, platí následující:

- Jestliže pozice není definována v pozičních datech (hodnota je 9E9), potom bude hodnota ignorována, jestliže osa je připojena a neaktivována. Ale jestliže osa je aktivována, výsledkem bude chyba.
- Jestliže pozice je definována v pozičních datech, třebaže osa není připojena, potom bude hodnota ignorována.

Nebude proveden žádný pohyb, ale nebude generována žádná chyba pro osu s platnými pozičními daty, jestliže osa není aktivována.

Jestliže pomocná osa běží v nezávislém režimu a nový pohyb má být proveden robotem a jeho pomocnými osami, potom poziční data pro pomocné osy v nezávislém režimu nesmí být 9E9. Data musí být libovolnou hodnotou, která není používána systémem.

Základní příklady

Následující příklady názorně ukazují datový typ `robtarget`:

Příklad 1

```
CONST robtarget p15 := [ [600, 500, 225.3], [1, 0, 0, 0], [1, 1,
     0, 0], [ 11, 12.3, 9E9, 9E9, 9E9, 9E9] ];
```

Pozice `p15` je definována takto:

- Pozice robotu: `x` = 600, `y` = 500 a `z` = 225.3 mm v souřadném systému objektu.
- Orientace nástroje ve stejném směru jako souřadný systém objektu.
- Konfigurace osy robotu: osy 1 a 4 v pozici 90-180°, osa 6 v pozici 0-90°.
- Pozice pomocných logických os a a b vyjádřená ve stupních nebo mm (podle typu osy). Osy c až f nejsou definovány.

Příklad 2

```
VAR robtarget p20;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.67 robtarget - Poziční data

RobotWare - OS

Pokračování

```
...
p20 := CRobT(\Tool:=tool\wobj:=wobjØ);
p20 := Offs(p20,10,0,0);
```

Pozice `p20` je nastavena ke stejné pozici jako aktuální pozice robotu voláním funkce `CRobT`. Pozice je potom posunuta o `10` mm ve směru x.

Konstrukce

```
< dataobject of robtarget >
    < trans of pos >
      < x of num >
      < y of num >
      < z of num >
    < rot of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
    < robconf of confdata >
      < cf1 of num >
      < cf4 of num >
      < cf6 of num >
      < cfx of num >
    < extax of extjoint >
      < eax_a of num >
      < eax_b of num >
      < eax_c of num >
      < eax_d of num >
      < eax_e of num >
      < eax_f of num >
```

Související informace

Pro informace o	Viz
Pohybové instrukce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Souřadné systémy*
Zpracování konfiguračních dat	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Konfigurace robotu*
Konfigurace pomocných os	*Application manual - Additional axes and stand alone controller*
Co je kvaternion?	*orient - Orientace na str 1542*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.68 sensor - Popisovač externího zařízení

Použití

`sensor` je popisovač k externímu zařízení, ke kterému bude provedeno připojení.

Popis

Popisovač pro zařízení na úrovni RAPID je uzavřen v záznamovém datovém typu `sensor`. Drží informace o senzorovém zařízení, jako je id, chybový kód a stav komunikace senzoru.

Komponenty

`id`

Datový typ: `num`

Interní identifikátor zařízení, který bude nastaven při první operaci se zařízením z úrovně RAPID. (Zatím nebylo implementováno).

`error`

Datový typ: `num`

Parametr `error` je nastaven, když je nastaven parametr `state` na `STATE_ERROR`. Když `state` přechází od `STATE_ERROR` na `STATE_CONNECTED`, parametr `error` je nastaven na 0.

Číslo chyby	Chyba
0	Žádná chyba.
112600	Inicializace komunikačního rozhraní selhala.
112602	Chyba komunikačního rozhraní

`state`

Datový typ: `sensorstate`

Odráží aktuální stav komunikace zařízení.

Příklady

Příklad datového typu `sensor` je uveden dole.

Příklad 1

```
PERS sensor AnyDevice;
PERS robdata DataOut := [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
PERS sensdata DataIn :=
     ["No",[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
VAR num SampleRate:=64;
...
! Setup Interface Procedure
PROC RRI_Open()
  SiConnect AnyDevice;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, SampleRate;
  SiSetCyclic AnyDevice, DataOut, SampleRate;
ENDPROC
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.68 sensor - Popisovač externího zařízení

Robor Reference Interface

Pokračování

Při volání rutiny `RRI_Open` je nejprve otevřeno spojení k zařízení `AnyDevice`. Potom je spuštěn cyklický přenos rychlostí `SampleRate`.

Konstrukce

```
<dataobject of sensor>
  <id of num>
  <error of num>
  <state of sensorstate>
```

Související informace

Pro informace o	Viz
Založit spojení k externímu systému.	Kontrola informačních štítků
Blízké spojení k externímu systému.	Kontrola informačních štítků
Data registru pro cyklický přenos.	Kontrola informačních štítků
Objednat přenos cyklických dat.	*SiGetCyclic - Rozhraní senzoru se zacyklilo. na str 646*
Stav komunikace zařízení.	Kontrola informačních štítků
Robot Reference Interface	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.69 sensorstate - Stav komunikace zařízení

Použití

`sensorstate` se používá k zastoupení aktuálního stavu komunikace zařízení.

Popis

Konstanta `sensorstate` se používá k reflektování aktuálního stavu komunikace zařízení. Může se použít z RAPIDu k vyhodnocení stavu spojení se senzorem.

Předdefinovaná data

Následující symbolické konstanty datového typu `sensorstate` jsou předdefinovány a mohou se používat k vyhodnocení, jaký je stav komunikace zařízení.

Konstanta	Hodnota
`STATE_ERROR`	`-1`
`STATE_UNDEFINED`	`0`
`STATE_CONNECTED`	`1`
`STATE_OPERATING`	`2`
`STATE_CLOSED`	`3`

Charakteristika

`sensorstate` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Založit spojení k externímu systému.	Kontrola informačních štítků
Blízké spojení k externímu systému.	Kontrola informačních štítků
Data registru pro cyklický přenos.	Kontrola informačních štítků
Objednat přenos cyklických dat.	*SiGetCyclic - Rozhraní senzoru se zacyklilo. na str 646*
Popisovač k externímu zařízení.	Kontrola informačních štítků
Robot Reference Interface	*Application manual - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.70 shapedata - Data tvaru světové zóny

Použití

`shapedata` se používá k popisu geometrie světové zóny.

Popis

Světové zóny mohou být definovány ve 4 různých geometrických tvarech:

- přímý box se všemi stranami souběžnými se světovým souřadným systémem a definovaný instrukcí `WZBoxDef`
- koule, definovaná instrukcí `WZSphDef`
- válec, souběžný s osou z světového souřadného systému a definovaný instrukcí `WZCylDef`
- oblast společného prostoru pro robot a/nebo externí osy, definovaná instrukcí `WZHomeJointDef` nebo `WZLimJointDef`

Geometrie světové zóny je definována jednou z předchozích instrukcí a činnost světové zóny je definována instrukcí `WZLimSup` nebo `WZDOSet`.

Základní příklady

Následující příklad názorně ukazuje datový typ `shapedata`:

Příklad 1

```
VAR wzstationary pole;
VAR wzstationary conveyor;
...
PROC ...
  VAR shapedata volume;
  ...
  WZBoxDef \Inside, volume, p_corner1, p_corner2;
  WZLimSup \Stat, conveyor, volume;
  WZCylDef \Inside, volume, p_center, 200, 2500;
  WZLimSup \Stat, pole, volume;
ENDPROC
```

`conveyor` je definován jako box a dohled pro tuto oblast je aktivován. `pole` je definován jako válec a dohled této zóny je také aktivován. Jestliže robot dosáhne jedné z těchto oblastí, pohyb je zastaven.

Charakteristika

`shapedata` je nehodnotový datový typ.

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Nastavení pohybu*
Definovat světovou zónu tvaru boxu	*WZBoxDef - Definovat světovou zónu tvaru box na str 1000*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Pro informace o	Viz
Definovat světovou zónu tvaru koule	*WZSphDef - Definovat světovou zónu tvaru koule na str 1025*
Definovat světovou zónu tvaru válce	*WZCylDef - Definovat světovou zónu tvaru válce na str 1002*
Definovat světovou zónu pro home spoje	*WZHomeJointDef - Definovat světovou zónu pro home spoje na str 1015*
Definovat světovou zónu pro limit spoje	*WZLimJointDef - Definovat světovou zónu pro omezení ve spojích na str 1018*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Aktivovat nastavení digitálního výstupu světové zóny	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.71 signalorigin - Popisuje počátek I/O signálu

Použití

`signalorigin` se používá k zastoupení celého čísla symbolickou konstantou.

Popis

Předdefinované symbolické konstanty typu `signalorigin` se mohou používat ke kontrole počátku I/O signálu. Je to určeno k používání při kontrole vratné hodnoty od funkce `GetSignalOrigin`.

Základní příklady

Následující příklad názorně ukazuje datový typ `signalorigin`:

Příklad 1

```
VAR signalorigin sigorig;
VAR string signalname;
...
sigorig := GetSignalOrigin(mydo, signalname);
IF sigorig = SIGORIG_NONE THEN
  TPWrite "The signal named "+ArgName(mydo)+" can not be used";
  Stop;
ELSEIF (sigorig = SIGORIG_CFG) OR (sigorig = SIGORIG_ALIAS) THEN
  SetDO mydo, 1;
  ...
ELSE
  TPWrite "Unknown origin "+ValToStr(sigorig);
  Stop;
ENDIF
```

Počátek signálu bude uložen do proměnné `sigorig`.

Předdefinovaná data

Následující konstanty typu `signalorigin` jsou předdefinovány:

Vratná hodnota (Vrátit hodnotu)	Symbolická konstanta	Komentář
0	`SIGORIG_NONE`	Proměnná I/O signálu je deklarována v RAPIDu a nemá žádnou vazbu alias.
1	`SIGORIG_CFG`	Signál je konfigurován v I/O konfiguraci.
2	`SIGORIG_ALIAS`	Proměnná I/O signálu je deklarována v RAPIDu a má vazbu alias na I/O signál konfigurovaný v I/O konfiguraci.

Charakteristika

`signalorigin` je datový typ alias pro `num` a tudíž následně zdědí jeho vlastnosti.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Získávání informací o počátku I/O signálu	*GetSignalOrigin - Získat informaci o původu I/O signálu na str 1180*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.72 signalxx - Digitální a analogové signály

Použití

Datové typy v `signalxx` se používají pro digitální a analogové vstupní a výstupní signály.

Jména signálů jsou definována v systémových parametrech a následně nemusí být definována v programu.

Popis

Datový typ	Použito pro
`signalai`	analogové vstupní signály
`signalao`	analogové výstupní signály
`signaldi`	digitální vstupní signály
`signaldo`	digitální výstupní signály
`signalgi`	skupiny digitálních vstupních signálů
`signalgo`	skupiny digitálních výstupních signálů

Proměnné typu `signalxo` obsahují pouze reference na signál. Hodnota se nastavuje pomocí instrukce, např. `DOutput`.

Proměnné typu `signalxi` obsahují referenci na signál, stejně tak jako možnost vyhledat hodnotu přímo v programu, jestliže je použita v kontextu hodnoty.

Hodnota vstupního signálu se může číst přímo v programu, například:

```
! Digital input
IF di1 = 1 THEN ...

! Digital group input
IF gi1 = 5 THEN ...

! Analog input
IF ai1 > 5.2 THEN ...
```

Může se také používat v přiděleních, např.:

```
VAR num current_value;

! Digital input
current_value := di1;

! Digital group input
current_value := gi1;

! Analog input
current_value := ai1;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Data typu `signalxx` nemusí být definována v programu. Nicméně, pokud jsou, bude zobrazena chybová zpráva, jakmile je vykonávána instrukce nebo funkce, která odkazuje k tomuto signálu. Datový typ může, na druhé straně, být použit jako parametr při deklarování rutiny.

Předdefinovaná data

K signálům definovaným v systémových parametrech je vždy přístup z programu pomocí předdefinovaných proměnných signálu (instalovaná data). Nicméně, mělo by být poznamenáno, že když jsou definována jiná data se stejným jménem, tyto signály není možné použít.

Charakteristika

`signalxx` je polohodnotový datový typ, který umožňuje operace orientované na hodnotu.

Řešení chyb

Následující odstranitelné chyby vznikají v chybném obslužném programu a tam je možné je řešit. Systémová proměnná `ERRNO` bude nastavena na:

`ERR_NO_ALIASIO_DEF` jestliže proměnná signálu je proměnná deklarovaná v RAPIDu. Nebyla připojena k I/O signálu definovanému v I/O konfiguraci s instrukcí `AliasIO`.

`ERR_NORUNUNIT`, jestliže není žádný kontakt s jednotkou.

Související informace

Pro informace o	Viz
Souhrn instrukcí vstup/výstup	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Vstupní a výstupní signály*
Funkčnost Vstup/Výstup všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Principy I/O*
Konfigurace I/O	*Technická referenční příručka - Systémové parametry*
Vlastnosti nehodnotových datových typů	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.73 socketdev - Socketové zařízení

Použití

`socketdev` (*socket device*) se používá ke komunikaci s jinými počítači na síti nebo mezi úlohami RAPID.

Popis

Socketové zařízení je odkaz na komunikační linku k jinému počítači nebo síti.

Základní příklady

Následující příklad názorně ukazuje datový typ `socketdev`:

Příklad 1

```
VAR socketdev socket1;
```

Proměnná `socket1` je definována a může se používat v socketovém příkazu, např. `SocketCreate`.

Omezení

Jakýkoliv počet socketů může být deklarován, ale je možné používat současně jen 32 socketů.

Charakteristika

`socketdev` je nehodnotový datový typ.

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Vytvoření nové zásuvky	*SocketCreate - Vytvořit nový socket na str 664*
Vlastnosti nehodnotových datových typů	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.74 socketstatus - Komunikační status socketu

Použití

`socketstatus` se používá k zastoupení komunikace socketu.

Popis

Status socketu je získán s funkcí `SocketGetStatus` a může se používat pro kontrolu toku programu nebo účely ladění.

Základní příklady

Následující příklad názorně ukazuje datový typ `socketstatus`:

Příklad 1

```
VAR socketdev socket1;
VAR socketstatus state;
...
SocketCreate socket1;
state := SocketGetStatus( socket1 );
```

Status socketu `SOCKET_CREATED` bude uložen do proměnné state.

Předdefinovaná data

Následující konstanty typu `socketstatus` jsou předdefinovány:

Konstanta RAPID	Hodnota	Socket je...
`SOCKET_CREATED`	1	Vytvořeno
`SOCKET_CONNECTED`	2	Klient je připojen ke vzdálenému hostiteli
`SOCKET_BOUND`	3	Server je navázán k lokální adrese a portu
`SOCKET_LISTENING`	4	Server poslouchá příchozí spojení
`SOCKET_CLOSED`	5	Zavřeno

Charakteristika

`socketstatus` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Socketová komunikace všeobecně	*Application manual - Controller software IRC5*
Získat status socketu	*SocketGetStatus - Získat aktuální stav socketu na str 1318*
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.75 speeddata - Rychlostní data

Použití

`speeddata` se používá k určení rychlosti, kterou se pohybuje robot a externí osy.

Description

Rychlostní data definují rychlost:

- jakou se pohybuje středový bod nástroje,
- rychlost reorientace nástroje,
- jakou se pohybují lineární nebo rotační externí osy.

Jestliže je kombinováno několik různých typů pohybu, jedna z rychlostí často omezuje všechny pohyby. Rychlost ostatních pohybů bude snížena tak, aby všechny pohyby ukončily provádění ve stejnou dobu.

Rychlost je často omezena činností robotu. Ta se mění podle typu robotu a dráhy pohybu.

Komponenty

`v_tcp`

velocity tcp

Datový typ:`num`

Rychlost středového bodu nástroje (TCP) v mm/s.

Jestliže je použit stacionární nástroj nebo koordinované externí osy, rychlost je určena podle pracovního objektu.

`v_ori`

velocity orientation

Datový typ: `num`

Rychlost reorientace TCP vyjádřená ve stupních/s.

Jestliže je použit stacionární nástroj nebo koordinované externí osy, rychlost je určena podle pracovního objektu.

`v_leax`

velocity linear external axes

Datový typ: `num`

Rychlost lineárních externích os v mm/s.

`v_reax`

velocity rotational external axes

Datový typ: `num`

Rychlost rotačních externích os ve stupních/s.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Základní příklady

Následující příklad názorně ukazuje datový typ `speeddata`:

Příklad 1

```
VAR speeddata vmedium := [ 1000, 30, 200, 15 ];
```

Rychlostní data `vmedium` jsou definována s následujícími rychlostmi:

- `1000` mm/s u TCP.
- `30` stupňů/s u reorientace nástroje.
- `200` mm/s u lineárních os.
- `15` mm/s u rotačních os.

```
vmedium.v_tcp := 900;
```

Rychlost TCP je změněna na `900` mm/s.

Omezení

U velmi pomalého pohybu by každý pohyb měl být dostatečně krátký, aby dal čas interpolace kratší než 240 sekund.

Předdefinovaná data

Číslo rychlostních dat je již definováno v systému.

Předdefinovaná rychlostní data, která se použijí pro pohyb robotu a externích os:

Název	Rychlost bodu TCP	Orientace	Lineární ext.osa	Rotační ext.osa
v5	5 mm/s	500°/s	5000 mm/s	1000°/s
v10	10 mm/s	500°/s	5000 mm/s	1000°/s
v20	20 mm/s	500°/s	5000 mm/s	1000°/s
v30	30 mm/s	500°/s	5000 mm/s	1000°/s
v40	40 mm/s	500°/s	5000 mm/s	1000°/s
v50	50 mm/s	500°/s	5000 mm/s	1000°/s
v60	60 mm/s	500°/s	5000 mm/s	1000°/s
v80	80 mm/s	500°/s	5000 mm/s	1000°/s
v100	100 mm/s	500°/s	5000 mm/s	1000°/s
v150	150 mm/s	500°/s	5000 mm/s	1000°/s
v200	200 mm/s	500°/s	5000 mm/s	1000°/s
v300	300 mm/s	500°/s	5000 mm/s	1000°/s
v400	400 mm/s	500°/s	5000 mm/s	1000°/s
v500	500 mm/s	500°/s	5000 mm/s	1000°/s
v600	600 mm/s	500°/s	5000 mm/s	1000°/s
v800	800 mm/s	500°/s	5000 mm/s	1000°/s
v1000	1000 mm/s	500°/s	5000 mm/s	1000°/s
v1500	1500 mm/s	500°/s	5000 mm/s	1000°/s
v2000	2000 mm/s	500°/s	5000 mm/s	1000°/s
v2500	2500 mm/s	500°/s	5000 mm/s	1000°/s

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.75 speeddata - Rychlostní data
RobotWare - OS
Pokračování

Název	Rychlost bodu TCP	Orientace	Lineární ext.osa	Rotační ext.osa
v3000	3000 mm/s	500°/s	5000 mm/s	1000°/s
v4000	4000 mm/s	500°/s	5000 mm/s	1000°/s
v5000	5000 mm/s	500°/s	5000 mm/s	1000°/s
v6000	6000 mm/s	500°/s	5000 mm/s	1000°/s
v7000	7000 mm/s	500°/s	5000 mm/s	1000°/s
vmax	*)	500°/s	5000 mm/s	1000°/s

*) Max. rychlost TCP pro použitý typ robotu a normální praktické hodnoty TCP. RAPID funkce `MaxRobSpeed` vrací stejnou hodnotu. Při použití extrémně velkých hodnot TCP v rámci nástroje potom vytvořte vlastní rychlostní data s vyšší rychlostí TCP, než která byla vrácena od `MaxRobSpeed`.

Předdefinovaná rychlostní data k použití pro posun rotačních externích os s instrukcí `MoveExtJ`.

Název	Rychlost bodu TCP	Orientace	Lineární ext.osa	Rotační ext.osa
vrot1	0 mm/s	0°/s	0 mm/s	1°/s
vrot2	0 mm/s	0°/s	0 mm/s	2°/s
vrot5	0 mm/s	0°/s	0 mm/s	5°/s
vrot10	0 mm/s	0°/s	0 mm/s	10°/s
vrot20	0 mm/s	0°/s	0 mm/s	20°/s
vrot50	0 mm/s	0°/s	0 mm/s	50°/s
vrot100	0 mm/s	0°/s	0 mm/s	100°/s

Předdefinovaná rychlostní data k použití pro posun lineárních externích os s instrukcí `MoveExtJ`.

Název	Rychlost bodu TCP	Orientace	Lineární ext.osa	Rotační ext.osa
vlin10	0 mm/s	0°/s	10 mm/s	0°/s
vlin20	0 mm/s	0°/s	20 mm/s	0°/s
vlin50	0 mm/s	0°/s	50 mm/s	0°/s
vlin100	0 mm/s	0°/s	100 mm/s	0°/s
vlin200	0 mm/s	0°/s	200 mm/s	0°/s
vlin500	0 mm/s	0°/s	500 mm/s	0°/s
vlin1000	0 mm/s	0°/s	1000 mm/s	0°/s

Konstrukce

```
< dataobject of speeddata >
  < v_tcp of num >
  < v_ori of num >
  < v_leax of num >
  < v_reax of num >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Pohyb/Rychlost všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Polohování během vykonávání programu*
Definování maximální rychlosti	*VelSet - Mění naprogramovanou rychlost na str 913*
Max. rychlost TCP pro tento robot	*MaxRobSpeed - Max rychlost robotu na str 1222*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.76 stoppointdata - Data stop bodu

Použití

`stoppointdata` se používá k určení, jak bude pozice ukončena, tj. jak blízko k naprogramované pozici musí být osy před pohybem dopředu k další pozici.

Popis

Pozice může být ukončena buď formou průjezdného bodu nebo stop bodu.

Průjezdný bod znamená, že naprogramované pozice není nikdy dosaženo. Zóna je určena v instrukci pro pohyb, definující rohovou dráhu. Namísto směřování k naprogramované pozici je směr pohybu utvářen do rohové dráhy, předtím než je pozice dosaženo. Viz datový typ `zonedata`.

Stop bod znamená, že robot a externí osy musí dosáhnout určené pozice předtím, než robot/externí osy pokračují s dalším pohybem. U robotu se má zato, že dosáhl stop bodu, když konvergenční kritéria bodu jsou splněna. Konvergenční kritéria jsou rychlost a pozice. Je také možné určit časovací kritéria. Ohledně stop bodu `fine` viz také datový typ `zonedata`.

Tři typy stop bodů je možné definovat pomocí `stoppointdata`.

- Typ stop bodu **v pozici** (in position) je definován jako procentní část konvergenčního kritéria (pozice a rychlost) pro předdefinovaný stop bod `fine`. Typ „v pozici“ také používá minimální a maximální čas. Robot čeká alespoň na minimální čas a nejdéle na maximální čas, aby kritéria pozice a rychlosti byla splněna.
- Typ stop bodu **stop čas** (stop time) vždy čeká ve stop bodu na daný čas.
- Typ stop bodu **následovaný čas** (follow time) je zvláštní typ stop bodu, který se používá ke koordinaci pohybů robotu s dopravníkem.

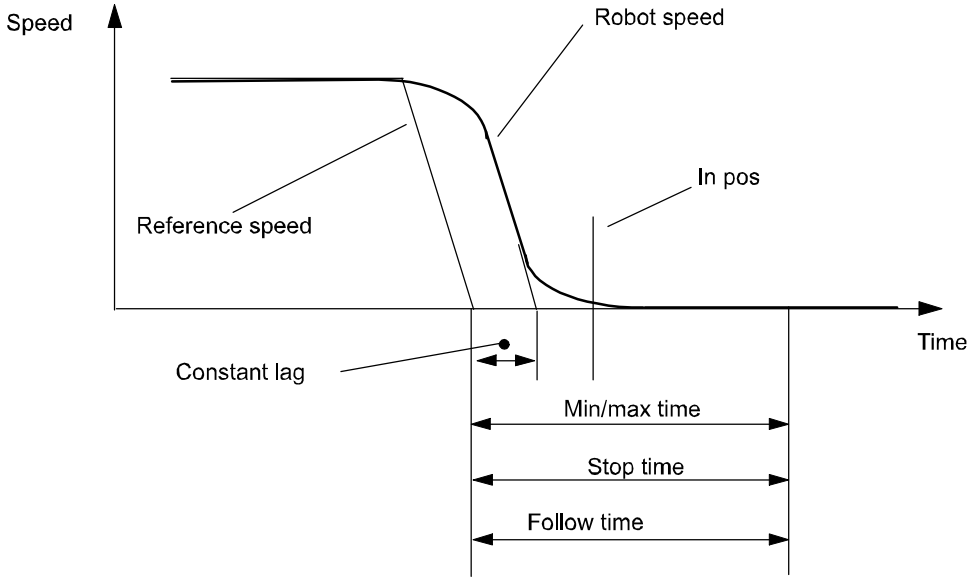
`stoppointdata` také stanoví, jak by měl být synchronizován pohyb s provedením RAPID. Jestliže pohyb je synchronizován, provedení RAPID čeká čeká na událost „in pos“, když je robot v pozici. Jestliže pohyb není synchronizován, provedení RAPID dostává událost „prefetch“ skoro půl sekundy předtím, než robot fyzicky dosáhne naprogramované pozice. Když vykonávání programu dostane událost „in pos“ nebo „prefetch“, pokračuje s další instrukcí. Když událost „prefetch“ přijde, robot má před sebou stále ještě dlouhou cestu. Když přijde událost „in pos“, robot je blízko naprogramované pozice.

U typu **stop čas** a **následovaný čas** začíná další instrukce své vykonávání ve stejné době jako stop čas a následovaný čas, resp. začínají odpočítávání. Ale u typu **v pozici** začíná další instrukce, když jsou splněna konvergenční kritéria.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Při používání pohybových instrukcí s argumentem `\Conc` není prováděna žádná synchronizace, takže aktuální pohybová instrukce bude hotová okamžitě.



xx0500002374

Na obrázku nahoře je popsáno ukončení stop bodů. Rychlost robotu se nesnižuje lineárně. Servo robotu je vždy před fyzickým robotem. Je uvedeno jako konstantní prodleva na obrázku nahoře. Konstantní prodleva je asi 0,1 sekundy. Časovací prvky `stoppointdata` využívají referenční rychlost jako spouštěč. Když je referenční rychlost nula, začíná měření času. Proto čas v časovacích prvcích vždy zahrnuje konstantní prodlevu. Následně nemá smysl používat hodnoty menší než je konstantní prodleva.

Komponenty

`type`

type of stop point

Datový typ: `stoppoint`

Následující tabulka definuje typ `stoppoint`.

`1 (inpos)`	Pohyb se ukončuje jako in-position typ stop bodu. Zapíná prvek `inpos` v `stoppointdata`. Zónová data v instrukci se nepoužívají, použijte `fine` nebo `z0`.
`2 (stoptime)`	Pohyb se ukončuje jako stop-time typ stop bodu. Zapíná prvek `stoptime` v `stoppointdata`. Zónová data v instrukci se nepoužívají, použijte `fine` nebo `z0`.
`3 (followtime)`	Pohyb se ukončuje jako follow-time typ jemného bodu do-pravníku. Zónová data v instrukci se používají, když robot opouští dopravník. Zapíná prvek `followtime` v `stoppointdata`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.76 stoppointdata - Data stop bodu
RobotWare - OS
Pokračování

Datový typ `stoppoint` je aliasový datový typ pro `num`. Používá se k výběru typu stop bodu a které datové prvky použít v `stoppointdata`. Předdefinované konstanty:

Hodnota	Symbolická konstanta	Komentář
1	`inpos`	Typové číslo In position
2	`stoptime`	Typové číslo Stop time
3	`fllwtime`	Typové číslo Follow time

`progsynch`

program synchronization

Datový typ: `bool`

Synchronizace s vykonáváním programu RAPID.

- `TRUE`: Pohyb je synchronizován s vykonáváním RAPID. Program nezačne vykonávat další instrukci před dosažením stop bodu.
- `FALSE`: Pohyb není synchronizován s vykonáváním RAPID. Program začne vykonávat další instrukci před dosažením stop bodu.

Při používání pohybových instrukcí s argumentem `\Conc` není prováděna žádná synchronizace nezávisle na datech `progsynch`, takže aktuální pohybová instrukce bude vždy hotová okamžitě.

`inpos.position`

position condition for TCP

Datový typ: `num`

Podmínka pozice (poloměr) pro TCP v procentech normálního stop bodu `fine`.

`inpos.speed`

speed condition for TCP

Datový typ: `num`

Podmínka rychlosti pro TCP v procentech normálního stop bodu `fine`.

`inpos.mintime`

minimum wait time

Datový typ: `num`

Minimální doba čekání v sekundách před in-position. Používá se, aby robot čekal alespoň po dobu určeného času v bodu. Max hodnota je 20,0 sekund.

`inpos.maxtime`

maximum wait time

Datový typ: `num`

Maximální doba čekání v sekundách kvůli splnění konvergenčních kritérií. Používá se pro zajištění, aby robot nezůstal zablokován v bodu, jestliže podmínky rychlosti a pozice jsou nastaveny příliš těsně. Max hodnota je 20,0 sekund.

`stoptime`

stop time

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Datový typ: `num`

Čas v sekundách, TCP stojí v klidu v pozici před spuštěním dalšího pohybu. Platný rozsah 0 - 20 s, rozlišení 0,001 s.

`followtime`

follow time

Datový typ: `num`

Čas v sekundách, TCP následuje dopravník. Platný rozsah 0 - 20 s, rozlišení 0,001 s.

`signal`

Datový typ: `string`

Rezervováno pro použití později.

`relation`

Datový typ: `opnum`

Rezervováno pro použití později.

`checkvalue`

Datový typ: `num`

Rezervováno pro použití později.

Základní příklady

Následující příklady názorně ukazují datový typ `stoppointdata`:

Inpos

```
VAR stoppointdata my_inpos := [ inpos, TRUE, [ 25, 40, 0.1, 5], 0,
     0, "", 0, 0];
MoveL *, v1000, fine \Inpos:=my_inpos, grip4;
```

Data stop bodu `my_inpos` jsou definována prostřednictvím následujících vlastností:

- Typ stop bodu je in-position, `inpos`.
- Stop bod bude synchronizován s vykonáváním programu RAPID, `TRUE`.
- Kritérium vzdálenosti stop bodu je 25 % vzdálenosti definované pro stop bod `fine`, 25.
- Kritérium rychlosti stop bodu je 40 % rychlosti definované pro stop bod `fine`, 40.
- Min doba čekání před konvergencí je 0,1 s, 0,1.
- Max doba čekání při konvergenci je 5 s, 5.

Robot se pohybuje k naprogramované pozici, dokud není splněno jedno z kritérií pozice a rychlosti.

```
my_inpos.inpos.position := 40;
MoveL *, v1000, fine \Inpos:=my_inpos, grip4;
```

Vzdálenostní kritérium stop bodu je nastaveno na 40 %.

Stoptime

```
VAR stoppointdata my_stoptime := [ stoptime, FALSE, [ 0, 0, 0, 0],
     1.45, 0, "", 0, 0];
MoveL *, v1000, fine \Inpos:=my_stoptime, grip4;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.76 stoppointdata - Data stop bodu
RobotWare - OS
Pokračování

Data stop bodu `my_stoptime` jsou definována prostřednictvím následujících vlastností:

- Typ stop bodu je stop-time, `stoptime`.
- Stop bod nebude synchronizován s vykonáváním programu RAPID, `FALSE`.
- Doba čekání v pozici je 1,45 s.

Robot se pohybuje k naprogramované pozici, dokud nepřijde událost prefetch. Vykonává se další instrukce RAPID. Jestliže je to pohybová instrukce, potom se robot zastaví na 1,45 sekundy před začátkem dalšího pohybu.

```
my_stoptime.stoptime := 6.66;
MoveL *, v1000, fine \Inpos:=my_stoptime, grip4;
```

Doba zastavení stop bodu je nastavena na 6,66 sekundy. Jestliže další instrukce RAPID je pohybová instrukce, robot se zastaví na 6,66 s.

Followtime

```
VAR stoppointdata my_followtime := [ fllwtime, TRUE, [ 0, 0, 0,
     0], 0, 0.5, "", 0, 0];
MoveL *, v1000, z10 \Inpos:=my_followtime, grip6\wobj:=conveyor1;
```

Data stop bodu `my_followtime` jsou definována prostřednictvím následujících vlastností:

- Typ stop bodu je follow-time, `fllwtime`.
- Stop bod bude synchronizován s vykonáváním programu RAPID, `TRUE`.
- Follow-time stop bodu jen 0,5 s, 0,5.

Robot bude následovat dopravník po dobu 0,5 s před jeho opuštěním se zónou 10 mm, `z10`.

```
my_followtime.followtime := 0.4;
```

Follow-time stop bodu je nastaven na 0,4 s.

Předdefinovaná data

Číslo dat stop bodu je již definováno v systému.

In-position stop body

Název	Progsynch	Pozice	Rych-lost	Min čas	Max čas	Stopti-me	Followti-me
inpos20	TRUE	20%	20%	0 s	2 s	-	-
inpos50	TRUE	50%	50%	0 s	2 s	-	-
inpos100	TRUE	100%	100%	0 s	2 s	-	-

(`inpos100` má stejné konvergenční kritérium jako stop bod `fine`)

Stop-time stop body

Název	Progsynch	Pozice	Rych-lost	Min čas	Max čas	Stopti-me	Followti-me
stoptime0_5	FALSE	-	-	-	-	0,5 s	-
stoptime1_0	FALSE	-	-	-	-	1,0 s	-
stoptime1_5	FALSE	-	-	-	-	1,5 s	-

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Follow-time stop body

Název	Progsynch	Pozice	Rych-lost	Min čas	Max čas	Stopti-me	Followti-me
fllwtime0_5	TRUE	-	-	-	-	-	0,5 s
fllwtime1_0	TRUE	-	-	-	-	-	1,0 s
fllwtime1_5	TRUE	-	-	-	-	-	1,5 s

Konstrukce

```
< data object of stoppointdata >
  < type of stoppoint >
  < progsynch of bool >
  < inpos of inposdata >
    < position of num >
    < speed of num >
    < mintime of num >
    < maxtime of num >
  < stoptime of num >
  < followtime of num >
  < signal of string >
  < relation of opnum >
  < checkvalue of num >
```

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Pohyby/Dráhy všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Polohování během vykonávání programu*
Stop nebo průjezdné body	*zonedata - Zónová data na str 1642*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.77 string (řetězec) - Řetězce

Použití

`string` se používá pro řetězce vlastností.

Popis

Řetězec vlastností se skládá z několika znaků (maximálně 80) vložených do uvozovek (""), např. "This is a character string".

Jestliže mají být vloženy do řetězce uvozovky (citace), musí být napsány dvakrát, např. "This string contains a ""character".

Jestliže mají být vložena do řetězce zpětná lomítka, musí být napsána dvakrát, např. "This string contains a \\ character".

Základní příklady

Následující příklad názorně ukazuje datový typ `string`:

Příklad 1

```
VAR string text;
...
text := "start welding pipe 1";
TPWrite text;
```

Text `start welding pipe 1` je zapsán na FlexPendant.

Omezení

Řetězec může mít 0 až 80 znaků; včetně extra otazníků nebo zpětných lomítek.

Řetězec může obsahovat každý znak uvedený v ISO 8859-1 (Latin-1), stejně tak jako kontrolní znaky (ne-ISO 8859-1 (Latin-1) s číselným kódem mezi 0-255).

Předdefinovaná data

Řada předdefinovaných řetězcových konstant je dostupná v systému a může se používat společně s funkcemi řetězce. Viz příklad `StrMemb`.

Název	Znaková sada
`STR_DIGIT`	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
`STR_UPPER`	<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï | 1)| Ñ | Ò | Ó | Ô | Õ | Ö | Ø | Ù | Ú | Û | Ü | 2) | 3)

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Název	Znaková sada
STR_LOWER	<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø | ù | ú | û | ü | 2) | 3) | ß | ÿ-
STR_WHITE	<blank character> ::=

1) Islandské písmeno eth.

2) Písmeno Y s ostrým akcentem.

3) Islandské písmeno thorn.

Následující konstanty jsou již definovány v systému:

```
CONST string diskhome := "HOME:";

! For old programs from S4C system
CONST string ram1disk := "HOME:";

CONST string disktemp := "TEMP:";

CONST string flp1 := "flp1:";

CONST string stSpace := " ";

CONST string stEmpty := "";
```

Související informace

Pro informace o	Viz
Operace používající řetězce	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Výrazy*
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Základní prvky*
Instrukce používající znakovou sadu	*StrMemb - Kontroluje, jestli znak patří do sady na str 1345*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.78 stringdig - Řetězec pouze s číslicemi

Použití

`stringdig` se používá k reprezentaci velkých kladných celých čísel v řetězci pouze s číslicemi.

Tento datový typ je zaveden, protože datový typ `num` nemůže zpracovat kladná celá čísla nad 8 388 608 s přesnou reprezentací.

Popis

`stringdig` se může skládat pouze z číslic 0 ... 9 vložených do uvozovek (""), např. "0123456789".

Datový typ `stringdig` může zpracovat kladná celá čísla do 4 294 967 295.

Základní příklady

Následující příklad názorně ukazuje datový typ `stringdig`:

Příklad 1

```
VAR stringdig digits1;
VAR stringdig digits2;
VAR bool flag1;
...
digits1 ="09000000";
digits2 = "9000001";
flag1 := StrDigCmp (digits1, LT, digits2);
```

Data `flag1` budou nastavena na `TRUE`, protože `09000000` je méně než `9000001`.

Charakteristika

`stringdig` je datový typ alias od `string` a následně zdědí většinu jeho vlastnosti.

Související informace

Pro informace o	Viz
Hodnoty řetězce	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Základní prvky*
Řetězce	*string (řetězec) - Řetězce na str 1596*
Numerické hodnoty	*num - Numerické hodnoty na str 1538*
Srovnávací operátor	*opnum - Srovnávací operátor na str 1541StrDigCmp - Porovnat dva řetězce pouze s číslicemi na str 1336*
Porovnat řetězce pouze s číslicemi	*StrDigCmp - Porovnat dva řetězce pouze s číslicemi na str 1336*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.79 supervtimeouts - Časové limity dohledu navázání spojení

Použití

`supervtimeouts` se používá k definování časových limitů pro dohled navázání spojení v CAP.

`supervtimeouts` je komponent `capdata` a definuje časové limity pro následující fáze dohledu navázání spojení v CAP:

- PRE
- END_PRE a START
- END MAIN a START_POST1
- END_POST1 a START_POST2
- END_POST2

Jestliže parametr je nastaven na 0, není zde žádný časový limit.

Komponenty

pre_cond

Datový typ: `num`

Časový limit (v sekundách) pro splnění podmínek fáze PRE.

start_cond

Datový typ: `num`

Časový limit (v sekundách) pro splnění podmínek fáze END_PRE a START.

end_main_cond

Datový typ: `num`

Časový limit (v sekundách) pro splnění podmínek fáze END_MAIN a START_POST1.

end_post1_cond

Datový typ: `num`

Časový limit (v sekundách) pro splnění podmínek fáze END_POST1 a START_POST2.

end_post2_cond

Datový typ: `num`

Časový limit (v sekundách) pro splnění podmínek fáze END_POST2.

Syntaxe

```
< data object of supervtimeouts >
  < pre_cond of num >
  < start_cond of num >
  < end_main_cond of num >
  < end_post1_cond of num >
  < end_post2_cond of num >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.79 supervtimeouts - Časové limity dohledu navázání spojení

Continuous Application Platform (CAP)

Pokračování

Související informace

	Popsáno v:
Datový typ `capdata`	*capdata - CAP data na str 1450*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.80 switch - Optional parameters

Použití

`switch` se používá pro volitelné parametry.

Popis

Speciální typ `switch` může být přidělen (pouze) volitelným parametrům a poskytuje prostředky pro použití přepínacích argumentů, to znamená argumentů, které jsou určeny pouze jmény (nikoliv hodnotami). Hodnotu není možné přenést na parametr přepínače. Jediný způsob, jak použít parametr přepínač, je kontrola jeho přítomnosti pomocí předdefinované funkce `Present`.

Základní příklady

Následující příklad názorně ukazuje datový typ `switch`:

Příklad 1

```
PROC my_routine(\switch on | \switch off)
....
IF Present (off) THEN
....
ENDIF
ENDPROC
```

Podle toho, jaké argumenty používá ten, kdo volá `my_routine`, může být kontrolován tok programu.

Charakteristika

`switch` je nehodnotový datový typ a nemůže se používat v operacích orientovaných na hodnotu.

Související informace

Pro informace o	Viz
Parametry	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Rutiny*
Jak kontrolovat, jestli je přítomen volitelný parametr	*Present - Otestovat, jestli je použit volitelný parametr na str 1274*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.81 symnum - Symbolické číslo

Použití

`symnum` (*Symbolic Number*) se používá k zastoupení celého čísla symbolickou konstantou.

Popis

Konstanta `symnum` je určena k použití při kontrole vracené hodnoty od funkcí `OpMode` a `RunMode`.

Základní příklady

Následující příklad názorně ukazuje datový typ `symnum`:

Příklad 1

```
IF RunMode() = RUN_CONT_CYCLE THEN
..
ELSE
..
ENDIF
```

Předdefinovaná data

Následující symbolické konstanty datového typu `symnum` jsou předdefinovány a mohou se používat ke kontrole vracených hodnot od funkcí `OpMode` a `RunMode`.

Hodnota	Symbolická konstanta	Komentář
0	`RUN_UNDEF`	Nedefinovaný provozní režim
1	`RUN_CONT_CYCLE`	Trvalý nebo cyklický provozní režim
2	`RUN_INSTR_FWD`	Instrukce pro režim běhu dopředu
3	`RUN_INSTR_BWD`	Instrukce pro režim běhu dozadu
4	`RUN_SIM`	Simulovaný provozní režim
5	`RUN_STEP_MOVE`	Pohybové instrukce v režimu běhu dopředu a logické instrukce v trvalém režimu běhu

Hodnota	Symbolická konstanta	Komentář
0	`OP_UNDEF`	Nedefinovaný provozní režim
1	`OP_AUTO`	Automatizovaný provozní režim
2	`OP_MAN_PROG`	Ruční provozní režim max. 250 mm/s
3	`OP_MAN_TEST`	Ruční provozní režim s plnou rychlostí, 100 %

Charakteristika

`Symnum` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.82 syncident - Identita pro bod synchronizace

Použití

`syncident` (*synchronization identity*) se používá k určení jména bodu synchronizace. Jméno bodu synchronizace bude jménem (identitou) deklarovaných dat typu `syncident`.

Popis

`syncident` se používá k identifikaci bodu v programu, kde aktuální programová úloha bude čekat na spolupracující programové úlohy, aby dosáhly stejného bodu synchronizace.

Jméno dat (identita) typu `syncident` musí být stejné ve všech spolupracujících programových úlohách.

Datový typ syncident se používá v instrukcích `WaitSyncTask`, `SyncMoveOn` a `SyncMoveOff`.

Základní příklady

Následující příklad názorně ukazuje datový typ `syncident`:

Příklad 1

Příklad programu v programové úloze `ROB1`

```
PERS tasks task_list{3} := [ ["STN1"], ["ROB1"], ["ROB2"] ];
VAR syncident sync1;

WaitSyncTask sync1, task_list;
```

Při vykonávání instrukce `WaitSyncTask` v programové úloze `ROB1` bude vykonávání v této programové úloze čekat, až ostatní programové úlohy `STN1` a `ROB2` dosáhnou jejich odpovídající `WaitSyncTask` se stejným synchronizačním (potkávacím) bodem `sync1`.

Konstrukce

`syncident` je nehodnotový datový typ.

Související informace

Pro informace o	Viz
Určit spolupracující programové úlohy	*tasks - Programové úlohy RAPID na str 1607*
Čekejte na bod synchronizace s ostatními úlohami	*WaitSyncTask - Čekat v bodu synchronizace na jiné programové úlohy na str 956*
Spustit koordinované synchronizované pohyby	*SyncMoveOn - Spustit koordinované synchronizované pohyby na str 758*
Ukončit koordinované synchronizované pohyby	*SyncMoveOff - Ukončit koordinované synchronizované pohyby na str 752*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.83 System data - Aktuální nastavení systémových dat RAPID

Použití

`System data` zrcadlí aktuální nastavení systémových dat RAPID, jako je nastavení pohybu aktuálního modelu, aktuální číslo obnovy po chybě `ERRNO`, aktuální číslo přerušení `INTNO` atd.

K těmto datům je možný přístup a čtení z programu. Mohou se použít pro čtení aktuálního statutu, např. aktuálního posunu programu.

`C_MOTSET`

Proměnná `C_MOTSET` datového typu `motsetdata` zrcadlí aktuální nastavení pohybu:

Popis	Datový typ	Změnil:	Viz část *System data - Aktuální nastavení systémových dat RAPID na str 1604*.
Aktuální nastavení pohybu, tj.:	`motsetdata`	Instrukce	*motsetdata - Data nastavení pohybu na str 1533*
Potlačení rychlosti a max rychlost		`VelSet`	*VelSet - Mění naprogramovanou rychlost na str 913*
Potlačení zrychlení		`AccSet`	*AccSet - Omezuje zrychlení na str 19*
Pohyby kolem singulárních bodů		`SingArea`	*SingArea - Definuje interpolaci kolem singulárních bodů na str 648*
Kontrola lineární konfigurace Kontrola společné konfigurace		`ConfL` `ConfJ`	*ConfJ - Kontroluje konfiguraci během lineárního pohybu na str 130* *ConfJ - Kontroluje konfiguraci během pohybu spoje na str 128*
Rozlišení dráhy		`PathResol`	*PathResol - Potlačit rozlišení dráhy na str 473*
Dohled ladicího pohybu		`MotionSup`	*MotionSup - Deaktivuje/aktivuje dohled (supervizi) pohybu na str 349*
Snížení zrychlení/zpomalení TCP podél dráhy pohybu		`PathAccLim`	*PathAccLim - Omezit TCP zrychlení podél dráhy na str 454*
Modifikace orientace nástroje během kruhové interpolace		`CirPathMode`	*CirPathMode - Reorientace nástroje během kruhové dráhy na str 105*
Snížení zrychlení užitečné zátěže ve světovém souřadném systému		`WorldAccLim`	*WorldAccLim - Kontrolovat zrychlení ve světovém souřadném systému na str 978*

`C_PROGDISP`

Proměnná `C_PROGDISP` datového typu `progdisp` zrcadlí aktuální posun programu a ofset externích os:

Popis	Datový typ	Změnil:	Viz část *System data - Aktuální nastavení systémových dat RAPID na str 1604*.
Aktuální posun programu u os robotu	`progdisp`	Instrukce:	*progdisp - Posun programu na str 1557*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Popis	Datový typ	Změnil:	Viz část *System data - Aktuální nastavení systémových dat RAPID na str 1604*.
		`PDispSet`	*PDispSet - Aktivuje posun programu pomocí známého rámce na str 481*
		`PDispOn`	*PDispOn - Aktivuje posun programu na str 477*
		`PDispOff`	*PDispOff - Deaktivuje posun programu na str 476*
Aktuální ofset externích os		`EOffsSet`	*EOffsSet - Aktivuje ofset pro pomocné osy pomocí známých hodnot na str 199*
		`EOffsOn`	*EOffsOn - Aktivuje ofset pro pomocné osy na str 197*
		`EOffsOff`	*EOffsOff - Deaktivuje ofset pro pomocné osy na str 196*

`ERRNO`

Proměnná `ERRNO` datového typu `errnum` zrcadlí aktuální číslo obnovy po chybě:

Popis	Datový typ	Změnil:	Viz část *System data - Aktuální nastavení systémových dat RAPID na str 1604*.
Poslední chyba, která vznikla	`errnum`	Systém	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Obnova po chybě* *intnum - Identita přerušení na str 1516*

`INTNO`

Proměnná `INTNO` datového typu `intnum` zrcadlí aktuální číslo přerušení:

Popis	Datový typ	Změnil:	Viz část *System data - Aktuální nastavení systémových dat RAPID na str 1604*.
Poslední přerušení, která vzniklo	`intnum`	Systém	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Přerušení* *intnum - Identita přerušení na str 1516*

`ROB_ID`

Proměnná `ROB_ID` datového typu `mecunit` obsahuje reference k TCP robotu (pokud existuje) v aktuální programové úloze.

Popis	Datový typ	Změnil:	Viz část *System data - Aktuální nastavení systémových dat RAPID na str 1604*.
Reference k robotu (pokud existuje) v aktuální programové úloze. Vždy kontrolujte před použitím s `TaskRunRob ()`	`mecunit`	Systém	*mecunit - Mechanická jednotka na str 1531*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.84 taskid - Identifikace úlohy

Použití

`taskid` se používá k identifikaci dostupných programových úloh v systému.

Jména programových úloh jsou definována v systémových parametrech a následně nemusí být definována v programu.

Popis

Data typu `taskid` obsahují pouze reference k programové úloze.

Omezení

Data typu `taskid` nemusí být definována v programu. Datový typ, na druhé straně, je možné použít jako parametr při deklarování rutiny.

Předdefinovaná data

Do programových úloh definovaných v systémových parametrech je vždy přístup z programu (instalovaná data).

Pro všechny programové úlohy v systému budou dostupné předdefinované proměnné datového typu `taskid`. Identita proměnné bude "taskname"+"Id", např. pro úlohu `T_ROB1` bude identita proměnné `T_ROB1Id`, `T_ROB2` - `T_ROB2Id` atd.

Charakteristika

`taskid` je nehodnotový datový typ. To znamená, že data tohoto typu nedovolují operace orientované na hodnotu.

Související informace

Pro informace o	Viz
Ukládání programových modulů	*Save - Uložit programový modul na str 578*
Konfigurace programových úloh	*Technická referenční příručka - Systémové parametry*
Vlastnosti nehodnotových datových typů	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.85 tasks - Programové úlohy RAPID

Použití

`tasks` se používá k určení několika programových úloh RAPID.

Popis

Kvůli určení několika programových úloh RAPID může být jméno každé úlohy dáno jako řetězec. Pole datového typu `tasks` může potom držet jména všech úloh.

Tento seznam úloh se může potom použít v instrukcích `WaitSyncTask` a `SyncMoveOn`.

POZNÁMKA

Instrukce nahoře vyžadují, aby data byla definována jako systémově globální proměnné `PERS` dostupné ve všech spolupracujících úlohách.

Komponenty

Datový typ má následující komponenty.

`taskname`

Datový typ: `string`

Jméno programové úlohy RAPID určené v řetězci.

Základní příklady

Následující příklad názorně ukazuje datový typ `tasks`:

Příklad 1

Příklad programu v programové úloze `T_ROB1`

```
PERS tasks task_list{3} := [ ["T_STN1"], ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;

WaitSyncTask sync1, task_list;
```

Při vykonávání instrukce `WaitSyncTask` v programové úloze `T_ROB1` bude vykonávání v této programové úloze čekat, až ostatní programové úlohy `T_STN1` a `T_ROB2` dosáhnou jejich odpovídající `WaitSyncTask` se stejným synchronizačním (potkávacím) bodem `sync1`.

Konstrukce

```
<dataobject of tasks>
  <taskname of string>
```

Související informace

Pro informace o	Viz
Identita pro bod synchronizace	*syncident - Identita pro bod synchronizace na str 1603*
Čekejte na bod synchronizace s ostatními úlohami	*WaitSyncTask - Čekat v bodu synchronizace na jiné programové úlohy na str 956*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.85 tasks - Programové úlohy RAPID

Multitasking

Pokračování

Pro informace o	Viz
Spustit koordinované synchronizované pohyby	*SyncMoveOn - Spustit koordinované synchronizované pohyby na str 758*
Ukončit koordinované synchronizované pohyby	*SyncMoveOff - Ukončit koordinované synchronizované pohyby na str 752*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.86 testsignal - Testovací signál

Použití

Datový typ `testsignal` se používá při provádění testu pohybového systému robotu.

Popis

Řada předdefinovaných testovacích signálů je k dispozici v systému robotu. Datový typ `testsignal` je k dispozici kvůli zjednodušení programování instrukce `TestSignDefine`.

Základní příklady

Následující příklad názorně ukazuje datový typ `testsignal`:

Příklad 1

```
TestSignDefine 2, speed, Orbit, 2, 0;
```

Předdefinovaná konstanta `speed` se používá ke čtení aktuální rychlosti osy `2` na manipulátoru `orbit`.

Předdefinovaná data

Následující testovací signály pro osy externího manipulátoru jsou předdefinovány v systému. Veškerá data jsou v jednotkách SI a jsou naměřena na motorové straně osy.

Symbolická konstanta	Hodnota	Jednotka
`speed`	6	rad/s
`torque_ref`	9	Nm
`resolver_angle`	1	rad
`speed_ref`	4	rad/s
`dig_input1`	102	0 nebo 1
`dig_input2`	103	0 nebo 1

Charakteristika

`testsignal` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Definovat testovací signál	*TestSignDefine - Definovat testovací signál na str 776*
Přečíst testovací signál	*TestSignRead - Přečíst hodnotu testovacího signálu na str 1366*
Resetovat testovací signály	*TestSignReset - Resetovat všechny definice testovacích signálů na str 778*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.87 tooldata - Data nástroje

Použití

`tooldata` se používá k popisu vlastností nástroje, například, svařovací pistole nebo chapadla. Tyto vlastnosti jsou pozice a orientace středového bodu nástroje (TCP) a fyzické vlastnosti zatížení nástroje.

Jestliže nástroj je upevněn v prostoru (stacionární nástroj), data nástroje nejprve definují pozici a orientaci tohoto nástroje v prostoru, TCP. Potom popisují zátěž chapadla posunovaného robotem.

Popis

Data nástroje ovlivňují pohyby robotu následujícími způsoby:

- Středový bod nástroje (TCP) referuje k bodu, který bude splňovat provedení určené dráhy a rychlosti. Jestliže nástroj je reorientován nebo jestliže jsou použity koordinované externí osy, pouze tento bod bude sledovat požadovanou dráhu při naprogramované rychlosti.
- Jestliže je použit stacionární nástroj, naprogramovaná rychlost a dráha budou mít souvislost s pracovním objektem drženému robotem.
- Naprogramované pozice referují k pozici aktuálního TCP a orientaci ve vztahu k souřadnému systému nástroje. To znamená, že, například, nástroj byl vyměněn, protože byl poškozen, starý program je možné stále používat, jestliže souřadný systém nástroje je znovu definován.

Data nástroje se také používají při ručním posuvu (jogging) robotu k:

- Definovat TCP, který se nepohybuje, když je nástroj reorientován.
- Definovat souřadný systém nástroje, aby byl usnadněn pohyb nebo otáčení ve směrech souřadnic nástroje.

VAROVÁNÍ

Je důležité vždy definovat skutečné zatížení nástroje a pokud se používá, užitečné zatížení robota (např. uchopený kus). Nesprávné definice zátěžových dat mohou vést k přetížení mechanické konstrukce robota.

Uvedení nesprávných údajů může často vést k následujícím důsledkům:

- Nebude využita maximální kapacita robota
- Bude narušena přesnost dráhy s rizikem minutí
- Vznikne riziko přetížení mechanické konstrukce

Komponenty

`robhold`

robot hold

Datový typ: `bool`

Definuje, jestli robot drží nástroj nebo nikoliv:

- `TRUE`: Robot drží nástroj.
- `FALSE`: Robot nedrží nástroj, tj. stacionární nástroj.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

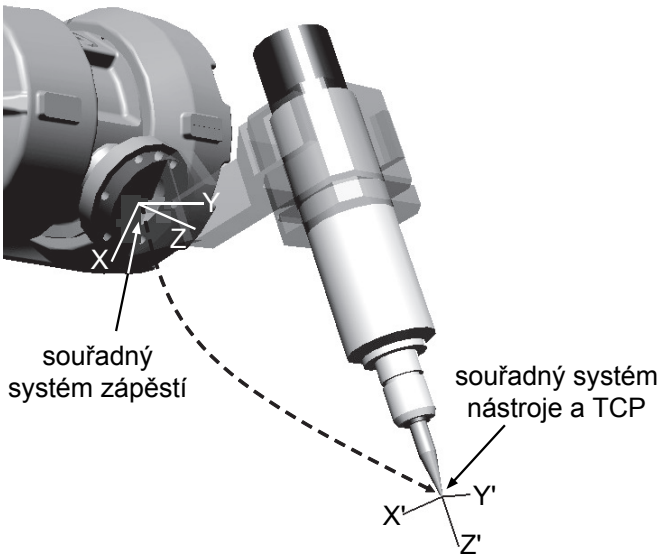
`tframe`

tool frame

Datový typ: `pose`

Souřadný systém nástroje, tj.:

- **Pozice TCP (x, y a z) v mm, vyjádřená v souřadném systému zápěstí (tool0) (viz obrázek dole).**
- **Orientace souřadného systému nástroje vyjádřená v souřadném systému zápěstí (viz obrázek dole).**



xx1100000517

Figure 3.3: Nástroj držený robotem

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.87 tooldata - Data nástroje
RobotWare - OS
Pokračování

POZNÁMKA

Jestliže je použit stacionární nástroj, rámec nástroje je definován ve vztahu ke světovému souřadnému systému.

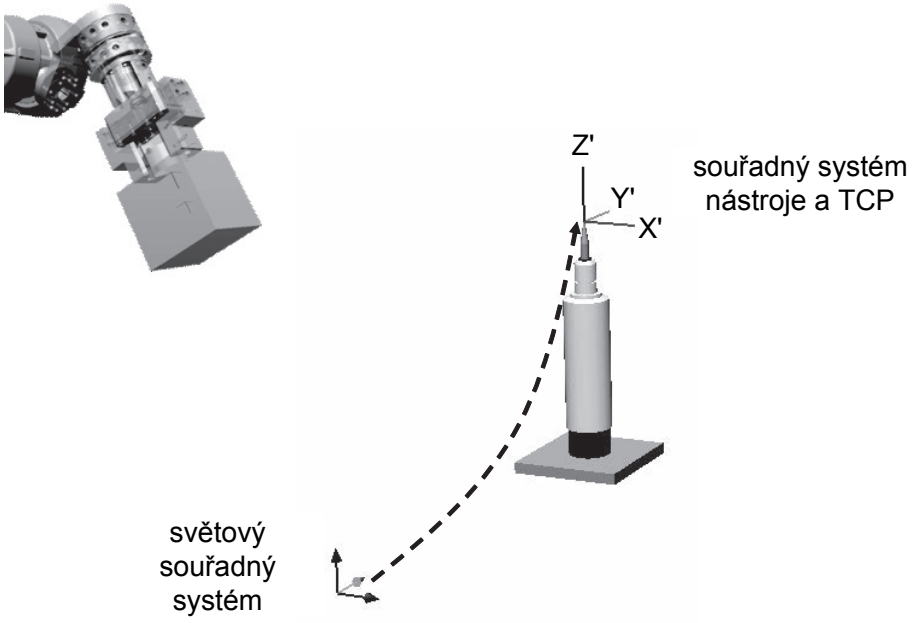


Figure 3.4: Stacionární nástroj

`tload`

tool load

Datový typ: `loaddata`

POZNÁMKA

Tato data se používají jak pro robot držící nástroj, tak pro stacionární nástroj. U robotu držícího nástroj data popisují zatížení nástroje. U stacionárního nástroje data popisují zatížení chapadla drženého robotem.

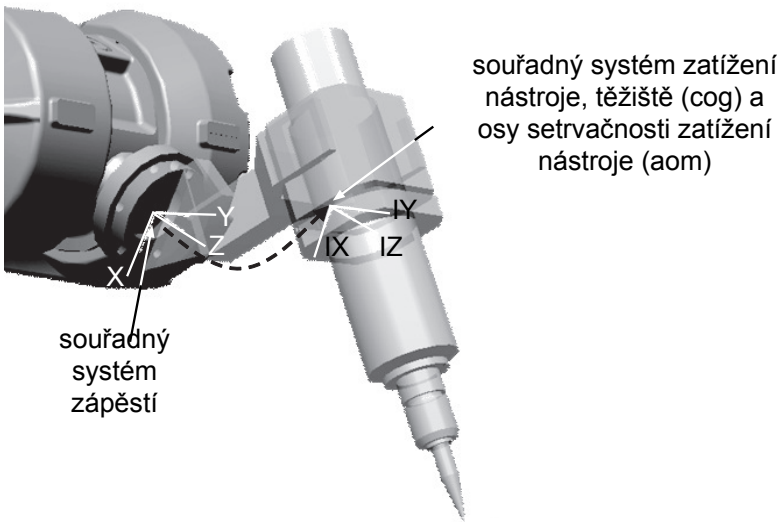
Nástroj držený robotem:

Zátěž nástroje, tj.:

- **Hmotnost nástroje v kg.**
- **Těžiště zátěže nástroje (x, y a z) v mm vyjádřené v souřadném systému zápěstí**
- **Orientace hlavních setrvačných os momentu nástroje vyjádřená v souřadném systému zápěstí.**
- **Momenty setrvačnosti kolem setrvačných os momentu v kgm^2. Jestliže všechny setrvačné komponenty jsou definovány jako 0 kgm^2, s nástrojem je zacházeno jako s bodovou masou.**

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

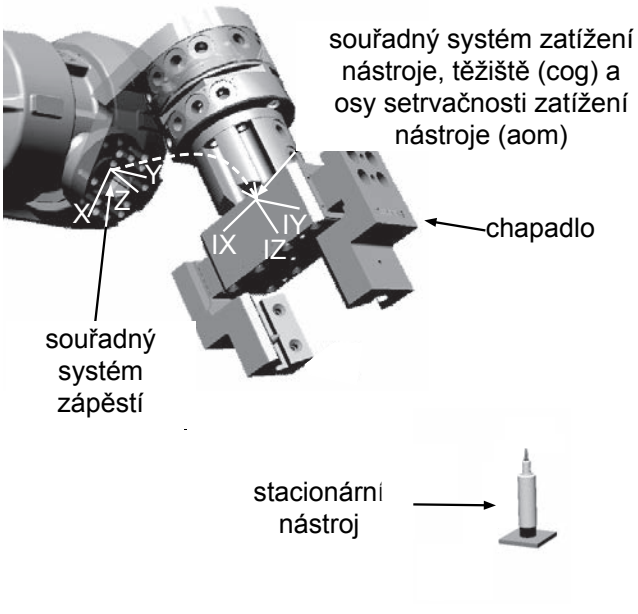


xx1100000519

Stacionární nástroj:

Zátěž chapadla držícího pracovní objekt:

- **Hmotnost posunutého chapadla v kg**
- **Těžiště posunutého chapadla (x, y a z) v mm vyjádřené v souřadném systému zápěstí**
- **Orientace hlavních setrvačných os momentu posunutého chapadla vyjádřená v souřadném systému zápěstí**
- **Momenty setrvačnosti kolem setrvačných os momentu v kgm^2. Jestliže všechny setrvačné komponenty jsou definovány jako 0 kgm^2, s chapadlem je zacházeno jako s bodovou masou.**



xx1100000520

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.87 tooldata - Data nástroje
RobotWare - OS
Pokračování

POZNÁMKA

Pouze zátěž nástroje/chapadla bude určena v `tooldata`. Užitečná zátěž zpracovaná chapadlem je připojována a odpojována instrukcí `GripLoad` a definována s `loaddata`.

Namísto používání instrukce `GripLoad` je možné definovat a používat různá `tooldata` pro *chapadlo s uchopených pracovním kusem* a *chapadlo bez pracovního kusu*.

Souhrn

Pozice a orientace TCP v `tooldata` jsou definovány v souřadném systému zápěstí pro nástroj držený robotem.

Pozice a orientace TCP v `tooldata` jsou definovány ve světovém souřadném systému pro stacionární nástroj .

Část `loaddata` v `tooldata` má ve všech případech vztah k souřadnému systému zápěstí, bez ohledu na fakt, jestli je použit nástroj držený robotem (pro popis nástroje) nebo stacionární nástroj (pro popis chapadla).

Základní příklady

Následující příklady názorně ukazují datový typ `tooldata`:

Příklad 1

```
PERS tooldata gripper := [ TRUE, [[97.4, 0, 223.1], [0.924, 0,
     0.383 ,0]], [5, [23, 0, 75], [1, 0, 0, 0], 0, 0, 0]];
```

Nástroj je popisován pomocí následujících hodnot:

- **Robot drží nástroj.**
- **TCP je umístěn v bodě 223,1 mm přímo ven z montážní příruby a 97,4 mm podél osy X souřadného systému zápěstí.**
- **Směry X' a Z' nástroje jsou otočeny o 45° ve vztahu ke směru Y v souřadném systému zápěstí.**
- **Hmotnost nástroje je 5 kg.**
- **Těžiště je umístěno v bodě 75 mm přímo ven z montážní příruby a 23 mm podél osy X souřadného systému zápěstí.**
- **Zátěž může být považována za bodovou masu, tj. bez jakéhokoliv momentu setrvačnosti.**

Příklad 2

```
gripper.tframe.trans.z := 225.2;
```

TCP nástroje, `gripper`, je upraven na 225,2 ve směru z.

Omezení

Data nástroje by měla být definována jako perzistentní proměnná (PERS) a nikoliv v rámci rutiny. Aktuální hodnoty jsou potom uloženy při ukládání programu a jsou vyhledány při načítání.

Argumenty typu data nástroje v jakékoliv pohybové instrukci by měly být pouze celým perzistentem (nikoliv elementem pole nebo komponentem záznamu).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Předdefinovaná data

Nástroj `tool0` definuje souřadný systém zápěstí s počátkem ve středu montážní příruby. K `tool0` je vždy možný přístup z programu, ale nikdy nemůže být změněn (je uložen v systémovém modulu BASE).

```
PERS tooldata tool0 := [ TRUE, [ [0, 0, 0], [1, 0, 0 ,0] ], [0.001,
     [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0] ];
```

Konstrukce

```
< dataobject of tooldata >
  < robhold of bool >
  < tframe of pose >
    < trans of pos >
      < x of num >
      < y of num >
      < z of num >
    < rot of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
  < tload of loaddata >
    < mass of num >
    < cog of pos >
      < x of num >
      < y of num >
      < z of num >
    < aom of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
    < ix of num >
    < iy of num >
    < iz of num >
```

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Souřadné systémy*
Definovat užitečnou zátěž pro roboty	*GripLoad - Definuje užitečnou zátěž pro robot na str 234*
Definice zátěžových dat	*loaddata - Zátěžová data na str 1523*
Definice dat pracovního objektu	*wobjdata - Data pracovního objektu na str 1634*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.88 tpnum - Číslo okna FlexPendantu

Použití

`tpnum` se používá k zastoupení čísla okna FlexPendantu symbolickou konstantou.

Popis

Konstanta `tpnum` je určena k používání v instrukci `TPShow`.

Základní příklady

Následující příklad názorně ukazuje datový typ `tpnum`:

Příklad 1

```
TPShow TP_LATEST;
```

Naposledy použité okno FlexPendantu před aktuálním oknem FlexPendantu bude aktivní po vykonání této instrukce.

Předdefinovaná data

Následující symbolická konstanta datového typu `tpnum` je předdefinována a může být použita v instrukci `TPShow`.

Hodnota	Symbolická konstanta	Komentář
2	TP_LATEST	Poslední použité okno FlexPendantu

Charakteristika

`tpnum` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Informace o	Viz
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*
Komunikace pomocí FlexPendantu	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Komunikace*
Okno přepínače na FlexPendantu	*TPShow - Okno přepínače na FlexPendantu na str 792*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.89 trapdata - Data přerušení pro aktuální TRAP

Použití

`trapdata` (*trap data*) se používá k držení dat přerušení, které způsobilo vykonání aktuální rutiny `TRAP`.

Používá se v `TRAP` rutinách generovaných instrukcí `IError`, před použitím instrukce `ReadErrData`.

Popis

Data typu `trapdata` reprezentují interní informace se vztahem k přerušení, které způsobilo vykonání aktuální trap rutiny. Obsah závisí na typu přerušení.

Základní příklady

Následující příklad názorně ukazuje datový typ `trapdata`:

Příklad 1

```
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
VAR trapdata err_data;
...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
ENDTRAP
```

Jestliže chyba je zachycena do trap rutiny `trap_err`, chybová doména, chybové číslo a typ chyby jsou uloženy do příslušných nehodnotových proměnných typu `trapdata`.

Charakteristika

`trapdata` je nehodnotový datový typ.

Související informace

Pro informace o	Viz
Souhrn přerušení	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Přerušení*
Více informací o správě přerušení	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Přerušení*
Nehodnotové datové typy	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*
Přikazuje přerušení na chyby	*IError - Přikazuje přerušení na chyby na str 246*
Získat data přerušení pro aktuální TRAP	*GetTrapData - Získat data přerušení pro aktuální TRAP na str 230*
Získává informace o chybě	*ReadErrData - Získává informace o chybě na str 527*
Advanced RAPID	*Specifikace produktu - Controller software IRC5*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.90 triggdata - Polohovací události, trigg

Použití

`triggdata` se používá k ukládání dat o polohovací události během pohybu robotu.

Polohovací událost může převzít formu nastavení výstupního signálu nebo běhu rutiny na určené pozici podél dráhy pohybu robotu.

Popis

Za účelem definování podmínek pro příslušná měření při polohovací události jsou použity proměnné typu `triggdata`. Obsahy dat proměnné jsou formovány v programu pomocí jedné z instrukcí `TriggIO`, `TriggEquip`, `TriggCheckIO`, `TriggInt`, `TriggSpeed` nebo `TriggRampAO` a jsou použity jednou z instrukcí `TriggL`, `TriggC` nebo`TriggJ`.

Základní příklady

Následující příklad názorně ukazuje datový typ `triggdata` :

Příklad 1

```
VAR triggdata gunoff;
TriggIO gunoff, 0,5 \DOp:=gun, 0;
TriggL p1, v500, gunoff, fine, gun1;
```

Digitální výstupní signál `gun` je nastaven na hodnotu `0`, když je TCP na pozici `0,5` mm před bodem `p1`.

Charakteristika

`triggdata` je nehodnotový datový typ.

Související informace

Pro informace o	Viz
Definice spouštěčů (triggs)	*TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814* *TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804* *TriggInt - Definuje přerušení se vztahem k pozici na str 820*
Použití spouštěčů (triggs)	*TriggL - Lineární pohyby robotu s událostmi na str 839* *TriggC - Kruhový pohyb robotu s událostmi na str 796* *TriggJ - Pohyby robotu podle osy s událostmi na str 831*
Vlastnosti nehodnotových datových typů	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.91 triggios - Positioning events, trigg

Použití

`triggios` se používá k uložení dat o polohovací události během pohybu robotu. Když je polohovací událost distribuována na určitou pozici na dráze, výstupní signál je nastaven na určenou hodnotu.

Popis

`triggios` se používá pro definování podmínek a činností pro nastavení digitálního výstupního signálu, skupiny digitálních výstupních signálů nebo analogového výstupního signálu na pevné pozici podél dráhy pohybu robotu.

Komponenty

`used`

Datový typ: `bool`

Definuje, jestli by měl být použit element pole nebo nikoliv.

`distance`

Datový typ: `num`

Definuje pozici na dráze, kde by mohla vzniknout událost I/O. Určeno jako vzdálenost v mm (kladná hodnota) od koncového bodu dráhy pohybu, jestliže komponent `start` je nastaven na FALSE.

`start`

Datový typ: `bool`

Nastaveno na TRUE, když vzdálenost začíná v bodě začátku pohybu namísto v koncovém bodě.

`equiplag`

Equipment Lag

Datový typ: `num`

Určit zpoždění pro externí zařízení v sek.

Pro kompenzaci zpoždění externího zařízení použijte kladnou hodnotu argumentu. Kladná hodnota znamená, že I/O signál je nastaven systémem robotu na určený čas, předtím, než TCP fyzicky dosáhne určené vzdálenosti ve vztahu k počátečnímu nebo koncovému bodu pohybu.

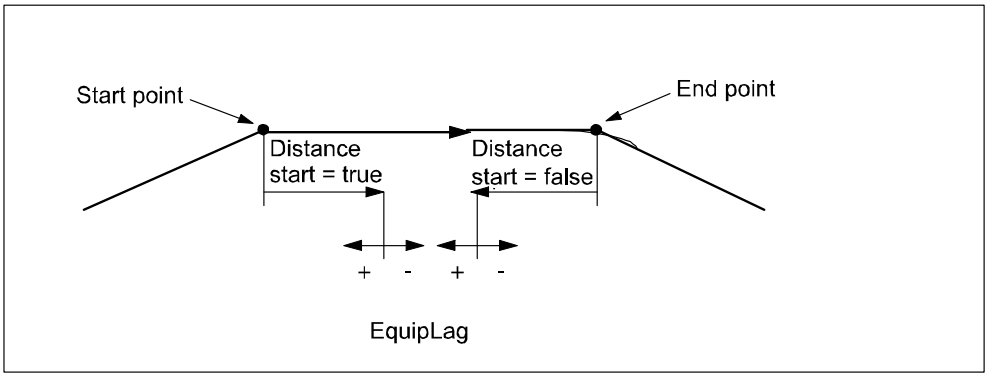
Záporná hodnota znamená, že I/O signál je nastaven systémem robotu na určený čas, po kterém TCP fyzicky přejde určenou vzdálenost ve vztahu k počátečnímu nebo koncovému bodu pohybu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.91 triggios - Positioning events, trigg
RobotWare - OS
Pokračování

Obrázek ukazuje použití komponentu `equiplag`.



xx0800000173

`signalname`

Datový typ: `string`

Jméno signálu, který bude změněn. Musí to být digitální výstupní signál, skupina digitálních výstupních signálů nebo analogový výstupní signál.

`setvalue`

Datový typ: `num`

Požadovaná hodnota výstupního signálu (v rámci povoleného rozsahu pro aktuální signál).

`xxx`

Datový typ: `num`

Komponent není právě používán. Doplněno, aby bylo možné přidávat funkčnost do příštích vydání při zachování kompatibility.

Příklady

Následující příklad názorně ukazuje datový typ `triggios`:

Příklad 1

```
VAR triggios gunon{1};

gunon{1}.used:=TRUE;
gunon{1}.distance:=3;
gunon{1}.start:=TRUE;
gunon{1}.signalname:="gun";
gunon{1}.equiplag:=0;
gunon{1}.setvalue:=1;

MoveJ p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData1:=gunon, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Signál `gun` je nastaven, když TCP je 3 mm za bodem `p1` .

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Konstrukce

```
<dataobject of triggios>
  <used of bool>
  <distance of num>
  <start of bool>
  <equiplag of num>
  <signalname of string>
  <setvalue of num>
  <xxx of num>
```

Související informace

Pro informace o	Viz
Polohovací události, trigg	*triggiosdnum - Positioning events, trigg na str 1622*
Lineární pohyby robotu s I/O událostmi	*TriggLIOs - Lineární pohyby robotu s I/O událostmi na str 854*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.92 triggiosdnum - Positioning events, trigg

Použití

`triggiosdnum` se používá k uložení dat o polohovací události během pohybu robotu. Když je polohovací událost distribuována na určitou pozici na dráze, výstupní signál je nastaven na určenou hodnotu.

Popis

`triggiosdnum` se používá pro definování podmínek a činností pro nastavení digitálního výstupního signálu, skupiny digitálních výstupních signálů nebo analogového výstupního signálu na pevné pozici podél dráhy pohybu robotu.

Komponenty

`used`

Datový typ: `bool`

Definuje, jestli by měl být použit element pole nebo nikoliv.

`distance`

Datový typ: `num`

Definuje pozici na dráze, kde by mohla vzniknout událost I/O. Určeno jako vzdálenost v mm (kladná hodnota) od koncového bodu dráhy pohybu, jestliže komponent `start` je nastaven na FALSE.

`start`

Datový typ: `bool`

Nastaveno na TRUE, když vzdálenost začíná v bodě začátku pohybu namísto v koncovém bodě.

`equiplag`

Equipment Lag

Datový typ: `num`

Určuje zpoždění pro externí zařízení v sekundách.

Pro kompenzaci zpoždění externího zařízení použijte kladnou hodnotu argumentu. Kladná hodnota znamená, že I/O signál je nastaven systémem robotu na určený čas, předtím, než TCP fyzicky dosáhne určené vzdálenosti ve vztahu k počátečnímu nebo koncovému bodu pohybu.

Záporná hodnota znamená, že I/O signál je nastaven systémem robotu na určený čas, potom, kdy TCP fyzicky přešel určenou vzdálenost ve vztahu k počátečnímu nebo koncovému bodu pohybu.

`signalname`

Datový typ: `string`

Jméno signálu, který bude změněn. Musí to být digitální výstupní signál, skupina digitálních výstupních signálů nebo analogový výstupní signál.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

`setvalue`

Datový typ: `dnum`

Požadovaná hodnota výstupního signálu (v rámci povoleného rozsahu pro aktuální signál).

`xxx`

Datový typ: `num`

Komponent není právě používán. Doplněno, aby bylo možné přidávat funkčnost do příštích vydání při zachování kompatibility.

Příklady

Následující příklad názorně ukazuje datový typ `triggiosdnum`**:**

Příklad 1

```
VAR triggiosdnum gunon{1};

gunon{1}.used:=TRUE;
gunon{1}.distance:=3;
gunon{1}.start:=TRUE;
gunon{1}.signalname:="go_gun";
gunon{1}.equiplag:=0;
gunon{1}.setvalue:=123456789;

MoveJ p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData3:=gunon, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Signál `go_gun` **je nastaven, když TCP je 3 mm za bodem** `p1` **.**

Konstrukce

```
<dataobject of triggiosdnum>
  <used of bool>
  <distance of num>
  <start of bool>
  <equiplag of num>
  <signalname of string>
  <setvalue of dnum>
  <xxx of num>
```

Související informace

Pro informace o	Viz
Polohovací události, trigg	*triggios - Positioning events, trigg na str 1619*
Lineární pohyby robotu s I/O událostmi	*TriggLIOs - Lineární pohyby robotu s I/O událostmi na str 854*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.93 triggmode - Režim činnosti trigg

Použití

`triggmode` se používá pro určení odlišných režimů činností při definování spouštěčů.

Popis

Konstanta `triggmode` je určena k používání při definování režimu pro instrukce používané pro definování spouštěčů.

Základní příklady

Následující příklady názorně ukazují datový typ `triggmode`:

Příklad 1

```
CONNECT intno1 WITH trap1;
TriggInt trigg1, Distance:=17, intno1 \Inhib:=inhibit
     \Mode:=TRIGG_MODE1;
TriggL p1, v500, trigg1, z50, gun1;
```

Rutina přerušení `trap1` je prováděna, když TCP je na pozici 17 mm před bodem `p1`, jestliže příznak perzistentní proměnné `inhibit` je `TRUE` (režim `TRIGG_MODE1` invertuje hodnotu přečtenou z příznaku `inhibit`).

Příklad 2

```
TriggEquip trigg1, 17, 0 \GOp:=go1, SetValue:=5 \Inhib:=inhibit
     \Mode:=TRIGG_MODE2;
TriggL p1, v500, trigg1, z50, gun1;
```

Jestliže perzistentní příznak `inhibit` je `FALSE`, když TCP je v pozici 17 mm před bodem `p1`, I/O signál `go1` je nastaven na hodnotu určenou v `SetValue`. Jestliže perzistentní proměnná `inhibit` je `TRUE`, není provedena žádná činnost (udržet hodnotu I/O signálu `go1`).

Příklad 3

```
TriggEquip trigg1, 17, 0 \GOp:=go1, SetValue:=0 \Inhib:=inhibit
     \InhibSetValue:=setDnum \Mode:=TRIGG_MODE3;
TriggL p1, v500, trigg1, z50, gun1;
```

Jestliže perzistentní příznak `inhibit` je `TRUE`, když TCP je v pozici 17 mm před bodem `p1`, I/O signál `go1` je nastaven na hodnotu přečtenou z `dnum` perzistentní proměnné `setDnum`. Jestliže `inhibit` je `FALSE`, není provedena žádná činnost (udržet hodnotu I/O signálu `go1`).

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Předdefinovaná data

Následující symbolické konstanty datového typu `triggmode` jsou předdefinovány a mohou se používat k určení různých režimů činnosti při definování spouštěčů.

Hodnota	Symbolická konstanta	Komentář
1	`TRIGG_MODE1`	Může být použito instrukcemi `TriggCheckIO`, `TriggEquip`, `TriggIO`, `TriggInt`, `TriggSpeed` a `TriggRampAO`. Invertovat hodnotu přečtenou z perzistentní proměnné použité ve volitelném argumentu `Inhib`.
2	`TRIGG_MODE2`	Může být použito instrukcemi `TriggEquip`, `TriggIO`, `TriggSpeed` a `TriggRampAO` při používání volitelného argumentu `Inhib`. Jestliže aktuální hodnota určeného příznaku použitého v `Inhib` je `TRUE` v pozici-čase pro nastavení I/O signálu, potom nebude určený I/O signál aktualizován (žádná činnost).
3	`TRIGG_MODE3`	Může se používat pouze společně s volitelnými argumenty `Inhib` a `InhibSetValue` pro instrukce `TriggEquip` a `TriggIO`. Argumenty `SetValue` a `SetDvalue` nejsou vůbec brány v úvahu. Jestliže aktuální hodnota určeného příznaku použitého v `Inhib` is `TRUE`, potom je I/O signál nastaven na hodnotu, kterou má perzistentní proměnná použitá ve volitelném argumentu `InhibSetValue` na pozici. Jestliže aktuální hodnota určeného příznaku použitého v `Inhib` je `FALSE`, I/O signál není aktualizován (žádná činnost).

Charakteristika

`triggmode` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Použití spouštěčů	*TriggL - Lineární pohyby robotu s událostmi na str 839* *TriggC - Kruhový pohyb robotu s událostmi na str 796* *TriggJ - Pohyby robotu podle osy s událostmi na str 831*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.93 triggmode - Režim činnosti trigg

RobotWare - OS

Pokračování

Pro informace o	Viz
Definice spouštěčů	*TriggEquip - Definovat pevnou pozici a časovou I/O událost na dráze na str 814* *TriggInt - Definuje přerušení se vztahem k pozici na str 820* *TriggIO - Definovat pevnou pozici nebo časovou I/O událost blízko stop bodu na str 825* *TriggRampAO - Definovat událost rampy AO pevné pozice na dráze na str 861* *TriggSpeed - Definuje proporční analogový výstup rychlosti TCP s událostí škály pevné pozice-času na str 868*
Definovat kontrolu I/O na pevné pozici	*TriggCheckIO - Definuje kontrolu IO v pevné pozici na str 804*
Uložení trigg dat	*triggdata - Polohovací události, trigg na str 1618*
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.94 triggstrgo - Positioning events, trigg

Použití

`triggstrgo`*(trigg stringdig group output)* se používá k uložení dat o polohovací události během pohybu robotu. Když je polohovací událost distribuována na určitou pozici na dráze, skupina digitálních výstupních signálů je nastavena na určenou hodnotu.

Popis

`triggstrgo` se používá pro definování podmínek a činností pro nastavení skupiny digitálních výstupních signálů na pevné pozici podél dráhy robotu.

Komponenty

`used`

Datový typ: `bool`

Definuje, jestli by měl být použit element pole nebo nikoliv.

`distance`

Datový typ: `num`

Definuje pozici na dráze, kde by mohla vzniknout událost I/O. Určeno jako vzdálenost v mm (kladná hodnota) od koncového bodu dráhy pohybu, jestliže komponent `start` je nastaven na FALSE.

`start`

Datový typ: `bool`

Nastaveno na TRUE, když vzdálenost začíná v bodě začátku pohybu namísto v koncovém bodě.

`equiplag`

Equipment Lag

Datový typ: `num`

Určit zpoždění pro externí zařízení v sek.

Pro kompenzaci zpoždění externího zařízení použijte kladnou hodnotu argumentu. Kladná hodnota znamená, že I/O signál je nastaven systémem robotu na určený čas, předtím, než TCP fyzicky dosáhne určené vzdálenosti ve vztahu k počátečnímu nebo koncovému bodu pohybu.

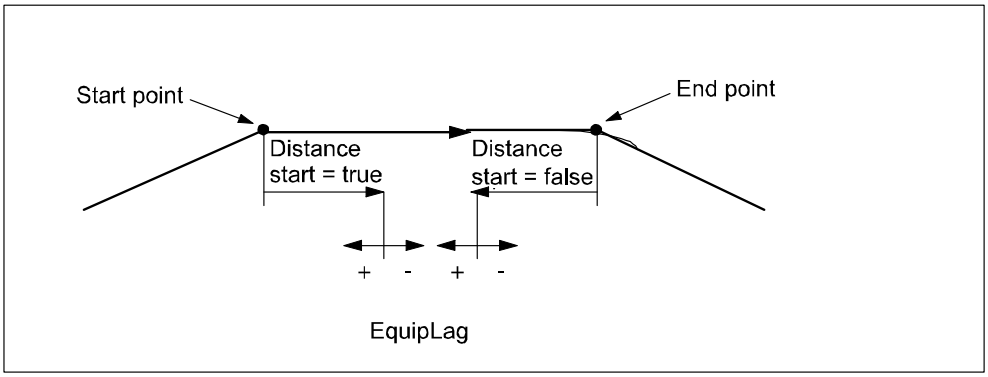
Záporná hodnota znamená, že I/O signál je nastaven systémem robotu na určený čas, po kterém TCP fyzicky přejde určenou vzdálenost ve vztahu k počátečnímu nebo koncovému bodu pohybu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.94 triggstrgo - Positioning events, trigg
RobotWare - OS
Pokračování

Obrázek ukazuje použití komponentu `equiplag`.



xx0800000173

`signalname`

Datový typ: `string`

Jméno signálu, který bude změněn. Musí to být jméno skupiny digitálních výstupních signálů.

`setvalue`

Datový typ: `stringdig`

Požadovaná hodnota výstupního signálu (v rámci povoleného rozsahu pro aktuální výstupní signál digitální skupiny). Použitím datového typu `stringdig` je umožněno používat hodnoty až 4294967295, což je max hodnota, jakou může skupina digitálních signálů mít (32 signálů v signálové skupině je max pro systém).

`xxx`

Datový typ: `num`

Komponent není právě používán. Doplněno, aby bylo možné přidávat funkčnost do příštích vydání při zachování kompatibility.

Příklady

Následující příklad názorně ukazuje datový typ `triggstrgo`:

Příklad 1

```
VAR triggstrgo gunon{1};

gunon{1}.used:=TRUE;
gunon{1}.distance:=3;
gunon{1}.start:=TRUE;
gunon{1}.signalname:="gun";
gunon{1}.equiplag:=0;
gunon{1}.setvalue:="4294967295";

MoveJ p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData2:=gunon, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Signál `gun` je nastaven na hodnotu 4294967295, když TCP je 3 mm za bodem `p1`.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Konstrukce

```
<dataobject of triggstrgo>
  <used of bool>
  <distance of num>
  <start of bool>
  <equiplag of num>
  <signalname of string>
  <setvalue of stringdig>
  <xxx of num>
```

Související informace

Pro informace o	Viz
Lineární pohyby robotu s I/O událostmi	*TriggLIOs - Lineární pohyby robotu s I/O událostmi na str 854*
Porovnat dva řetězce pouze s číslicemi	*StrDigCmp - Porovnat dva řetězce pouze s číslicemi na str 1336*
Aritmetické operace na datových typech stringdig	*StrDigCalc - Aritmetické operace s datovým typem stringdig na str 1333*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.95 tunetype - Typ servo ladění

Použití

`tunetype` se používá k zastoupení celého čísla symbolickou konstantou pro různé typy servo ladění.

Popis

Konstanta tunetype je určena k používání jako argument v instrukci `TuneServo`.

Základní příklady

Následující příklad názorně ukazuje datový typ `tunetype`:

Příklad 1

```
TuneServo MHA160R1, 1, 110 \Type:= TUNE_KP;
```

Předdefinovaná data

Následující symbolické konstanty datového typu `tunetype` jsou předdefinovány a mohou se používat jako argumenty pro instrukci `TuneServo`.

Hodnota	Symbolická konstanta	Komentář
0	TUNE_DF	Snížená přejetí
1	TUNE_KP	Ovlivňuje zisk kontroly pozice
2	TUNE_KV	Ovlivňuje zisk kontroly rychlosti
3	TUNE_TI	Ovlivňuje integrační dobu kontroly rychlosti
4	TUNE_FRIC_LEV	Ovlivňuje kompenzační úroveň tření
5	TUNE_FRIC_RAMP	Ovlivňuje kompenzační rampu tření
6	TUNE_DG	Snížená přejetí
7	TUNE_DH	Snižuje vibrace s těžkými náklady
8	TUNE_DI	Snižuje chyby dráhy
9	TUNE_DK	Pouze pro vnitřní potřebu ABB
10	TUNE_DL	Pouze pro vnitřní potřebu ABB

Charakteristika

`tunetype` je datový typ alias pro `num` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Datové typy všeobecně, datové typy alias	*Technická referenční příručka - Přehled RAPID*, sekce *Základní vlastnosti - Datové typy*
Použití datového typu tunetype	*TuneServo - Ladění serv na str 886*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.96 uishownum - Instance ID pro UIShow

Použití

`uishownum` je datový typ používaný pro parametr `InstanceId` v instrukci `UIShow`. Používá se k identifikaci náhledu na FlexPendantu.

Popis

Když se používá perzistentní proměnná typu `uishownum` s instrukcí `UIShow`, dostane konkrétní hodnotu identifikující náhled spuštěný na FlexPendantu. Tento perzistent je potom použit při všech manipulacích s tímto náhledem, jako je spuštění náhledu znovu, modifikace náhledu atd.

Příklady

Následující příklad názorně ukazuje datový typ `uishownum`:

Příklad 1

```
CONST string Name:="TpsViewMyAppl.gtpu.dll";
CONST string Type:="ABB.Robotics.SDK.Views.TpsViewMyAppl";
CONST string Cmd1:="Init data string passed to the view";
PERS uishownum myinstance:=0;
VAR num mystatus:=0;
...
! Launch one view of the application MyAppl
UIShow Name, Type \InitCmd:=Cmd1 \InstanceID:=myinstance
     \Status:=mystatus;
```

Kód nahoře spustí jeden náhled aplikace `MyAppl` s init příkazem `Cmd1`. Token použitý k identifikaci náhledu je uložen do parametru `myinstance`.

Charakteristika

`uishownum` je datový typ alias pro `num` a tudíž následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
UIShow	*UIShow - Ukázka uživatelského rozhraní na str 901*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.97 weavestartdata - spouštěcích data weave

Použití

`weavestartdata` se používá ke kontrole stacionárního weavingu během startu a restartu procesu v CAP.

`weavestartdata` je komponent `capdata` a definuje vlastnosti stacionárního weavingu při startu nebo restartu procesu CAP:

- jestli bude stacionární weaving při startu (`active`)
- šířka stacionárního weavingu (`width`)
- směr dráhy vzhledem ke směru (`dir`)
- kmitočet stacionárního weavingu (`cycle_time`)

Stacionární weaving používá vždy geometrický weaving s cik-cak vzorem, viz *capweavedata - Weavedata pro CAP na str 1463*.

Komponenty

active

Datový typ: `bool`

Hodnota	Popis
TRUE	Proveďte stacionární weaving při startu procesu CAP
FALSE	NEPROVÁDĚJTE stacionární weaving při startu procesu CAP

width

Datový typ: `num`

Definuje amplitudu stacionárního weavingu (mm).

dir

Datový typ: `num`

Definuje směr stacionárního weavingu vzhledem ke směru dráhy (stupně). Nula stupňů znamená weaving kolmý na dráhu a z-souřadnici nástroje.

cycle_time

Datový typ: `num`

Definuje celkový čas (v sekundách) kompletního cyklu stacionárního weavingu.

Syntaxe

```
< data object of weavestartdata >
  < active of bool >
  < width of num >
  < dir of num >
  < cycle_time of num >
```

Související informace

	Popsáno v:
Datový typ `capdata`	*capdata - CAP data na str 1450*

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.97 weavestartdata - spouštěcích data weave

Continuous Application Platform (CAP)

Pokračování

	Popsáno v:
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.98 wobjdata - Data pracovního objektu

Použití

`wobjdata` se používá k popisu pracovního objektu, který robot svařuje, zpracovává, pohybuje se v něm atd.

Popis

Jestliže jsou pracovní objekty definovány v polohovací instrukci, pozice bude založena na souřadnicích pracovního objektu. Výhody jsou následující:

- Jestliže jsou poziční data vkládána ručně, jako např. v offline programování, hodnoty mohou být často převzaty z výkresu.
- Programy mohou být znovu použity rychle po změnách v instalaci robotu. Jestliže, například, je posunut upevňovací prvek, musí být znovu definován pouze souřadný systém uživatele.
- Variace v tom, jak je pracovní objekt připojen, mohou být kompenzovány. Kvůli tomu, nicméně, bude třeba nějaký druh senzoru, aby pracovní objekt mohl být usazen do pozice.

Jestliže je použit stacionární nástroj nebo koordinované externí osy, pracovní objekt musí být definován, jelikož dráha a rychlost budou mít potom vztah k pracovnímu objektu namísto TCP.

Data pracovního objektu mohou být použita také pro ruční posuv (jogging):

- Robot může být joggován ve směrech pracovního objektu.
- Aktuální zobrazená pozice je založena na souřadném systému pracovního objektu.

Komponenty

`robhold`

robot hold

Datový typ: `bool`

Definuje, jestli robot v aktuální programové úloze drží pracovní objekt nebo nikoliv:

- `TRUE`: Robot drží pracovní objekt, tj. pomocí stacionárního nástroje.
- `FALSE`: Robot nedrží pracovní objekt, tj. robot drží nástroj.

`ufprog`

user frame programmed

Datový typ: `bool`

Definuje, jestli je použit pevný uživatelský souřadný systém nebo nikoliv:

- `TRUE`: Pevný uživatelský souřadný systém.
- `FALSE`: Pohyblivý uživatelský souřadný systém, tj. jsou použity koordinované externí osy. Používá se také v systému MultiMove v polokoordinovaném nebo synchonizovaném koordinovaném režimu.

`ufmec`

user frame mechanical unit

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Datový typ: `string`

Mechanická jednotka, se kterou jsou pohyby robotu koordinovány. Upřesněno pouze v případě pohyblivých uživatelských koordinovaných systémů (`ufprog` je `FALSE`).

Určit jméno mechanické jednotky definované v systémových parametrech, např. `orbit_a`.

`uframe`

user frame

Datový typ: `pose`

Uživatelský souřadný systém, tj. pozice aktuální pracovní plochy nebo upínadla (viz obrázek dole):

- Pozice počátku souřadného systému (x, y a z) v mm.
- Rotace souřadného systému vyjádřená jako kvaternion (q1, q2, q3, q4).

Jestliže robot drží nástroj, uživatelský souřadný systém je definován ve světovém souřadném systému (v souřadném systému zápěstí, jestliže se používá stacionární nástroj).

U pohyblivého uživatelském rámce (`ufprog` je `FALSE`), je uživatelský rámec nepřetržitě definován systémem.

`oframe`

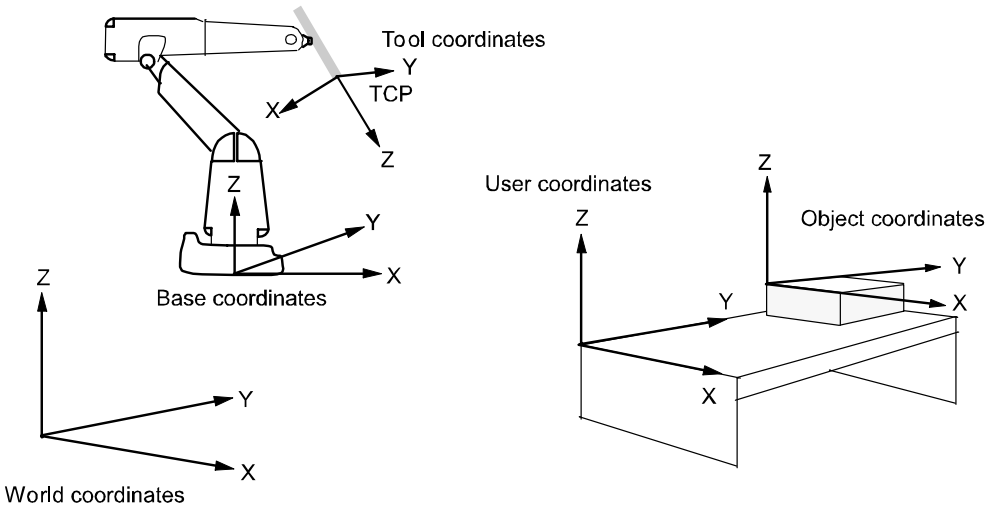
object frame

Datový typ: `pose`

Souřadný systém objektu, tj. pozice aktuálního pracovního objektu (viz obrázek dole):

- Pozice počátku souřadného systému (x, y a z) v mm.
- Rotace souřadného systému vyjádřená jako kvaternion (q1, q2, q3, q4).

Souřadný systém objektu je definován v uživatelském souřadném systému.



xx0500002369

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.98 wobjdata - Data pracovního objektu
RobotWare - OS
Pokračování

Základní příklady

Následující příklad názorně ukazuje datový typ `wobjdata`:

Příklad 1

```
PERS wobjdata wobj2 :=[ FALSE, TRUE, "", [ [300, 600, 200], [1, 0,
     0 ,0] ], [ [0, 200, 30], [1, 0, 0 ,0] ] ];
```

Pracovní objekt na obrázku nahoře je popsán pomocí následujících hodnot:

- Robot nedrží pracovní objekt.
- Je použit pevný uživatelský souřadný systém.
- Uživatelský souřadný systém není otočen a souřadnice jeho počátku jsou x = 300, y = 600 a z = 200 mm ve světovém souřadném systému.
- Souřadný systém objektu není otočen a souřadnice jeho počátku jsou x = `0`, y = 200 a z = 30 mm v uživatelském souřadném systému.

  ```
  wobj2.oframe.trans.z := 38.3;
  ```

- Pozice pracovního objektu `wobj2` je upravena na 38,3 ve směru z.

Omezení

Data pracovního objektu by měla být definována jako perzistentní proměnná (PERS) a nikoliv v rámci rutiny. Aktuální hodnoty jsou potom uloženy při ukládání programu a jsou vyhledány při načítání.

Argumenty typu data pracovního objektu v jakékoliv pohybové instrukci by měly být pouze celým perzistentem (nikoliv elementem pole nebo komponentem záznamu).

Předdefinovaná data

Data pracovního objektu `wobj0` jsou definována tak, že souřadný systém objektu se shoduje se světovým souřadným systémem. Robot nedrží pracovní objekt.

K `Wobj0` je možné vždy přistoupit z programu, ale nemůže být změněn (je uložen v systémovém modulu BASE).

```
PERS wobjdata wobj0 := [ FALSE, TRUE, "", [ [0, 0, 0], [1, 0, 0
     ,0] ], [ [0, 0, 0], [1, 0, 0 ,0] ] ];
```

Konstrukce

```
< dataobject of wobjdata >
  < robhold of bool >
  < ufprog of bool >
  < ufmec of string >
  < uframe of pose >
    < trans of pos >
      < x of num >
      < y of num >
      < z of num >
    < rot of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
< oframe of pose >
  < trans of pos >
    < x of num >
    < y of num >
    < z of num >
  < rot of orient >
    < q1 of num >
    < q2 of num >
    < q3 of num >
    < q4 of num >
```

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Souřadné systémy	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Souřadné systémy*
Koordinované externí osy	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Souřadné systémy*
Kalibrace externích os	*Application manual - Additional axes and stand alone controller* *Application manual - MultiMove*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.99 wzstationary - Data stacionární světové zóny

Použití

`wzstationary` (*world zone stationary*) se používá k identifikaci stacionární světové zóny a může se používat pouze v událostní rutině připojené k události POWER ON.

Světová zóna je dohlížena během pohybů robotu, jak během vykonávání programu, tak i během joggování. Jestliže TCP robotu dosáhne světové zóny nebo jestliže robot/externí osy dosáhnou světové zóny ve spojích, pohyb je zastaven nebo digitální výstupní signál je nastaven nebo resetován.

Popis

Světová zóna `wzstationary` je definována a aktivována instrukcí `WZLimSup` nebo `WZDOSet`.

`WZLimSup` nebo `WZDOSet` dává numerickou hodnotu proměnné nebo perzistentu datového typu `wzstationary`. Hodnota identifikuje světovou zónu.

Stacionární světová zóna je **vždy aktivní** ve stavu zapnutého motoru a vymaže ji pouze **Restart**. Není možné deaktivovat, aktivovat nebo vymazat stacionární světovou zónu přes instrukce RAPID.

Stacionární světové zóny by měly být aktivní od zapnutí napájení a měly by být definovány v událostní rutině POWER ON nebo polostatické úloze

Základní příklady

Následující příklad názorně ukazuje datový typ `wzstationary`:

Příklad 1

```
VAR wzstationary conveyor;
...
PROC ...
  VAR shapedata volume;
  ...
  WZBoxDef \Inside, volume, p_corner1, p_corner2;
  WZLimSup \Stat, conveyor, volume;
ENDPROC
```

`conveyor` je definován jako přímý box (objem pod pásem). Jestliže robot dosáhne tohoto objemu, pohyb je zastaven.

Omezení

Data `wzstationary` mohou být definována jako proměnná (**VAR**) nebo jako perzistent (**PERS**). Mohou být globální v úloze nebo lokální v modulu, ale nikoliv lokální v rutině.

Argumenty typu `wzstationary` by měly být pouze celými daty (nikoliv elementem pole nebo komponentem záznamu).

Init hodnota pro data typu `wzstationary` není kontrolním systémem používána. Když je potřeba použít perzistentní proměnnou a multitaskingovém systému,

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

nastavte init hodnotu na 0 v obou úlohách, např. `PERS wzstationary share_workarea := [0];`

Další příklady

Kompletní příklad můžete vidět v instrukci `WZLimSup`.

Charakteristika

`wzstationary` je datový typ alias od `wztemporary` a následně zdědí jeho vlastnosti.

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Světové zóny*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Dočasná světová zóna	*wztemporary - Data dočasné světové zóny na str 1640*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Aktivovat nastavení digitálního výstupu světové zóny	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.100 wztemporary - Data dočasné světové zóny

Použití

`wztemporary` (*world zone temporary*) se používá k identifikaci dočasné světové zóny a může se používat kdekoliv v programu RAPID pro každou pohybovou úlohu.

Světová zóna je dohlížena během pohybů robotu, jak během vykonávání programu, tak i během joggování. Jestliže TCP robotu dosáhne světové zóny nebo jestliže robot/externí osy dosáhnou světové zóny ve spojích, pohyb je zastaven nebo digitální výstupní signál je nastaven nebo resetován.

Popis

Světová zóna `wztemporary` je definována a aktivována instrukcí `WZLimSup` nebo `WZDOSet`.

`WZLimSup` nebo `WZDOSet` dává numerickou hodnotu proměnné nebo perzistentu datového typu `wztemporary`. Hodnota identifikuje světovou zónu.

Jakmile je definována a aktivována, dočasnou světovou zónu je možné deaktivovat pomocí `WZDisable`, aktivovat znovu pomocí `WZEnable` a vymazat pomocí `WZFree`.

Všechny dočasné světové zóny v pohybové úloze jsou automaticky vymazány a všechny datové objekty typu `wztemporary` v pohybové úloze jsou nastaveny na 0:

- když je nový program načten do motion (pohybové) úlohy
- při spuštění vykonávání programu od začátku v motion (pohybové) úloze

Základní příklady

Následující příklad názorně ukazuje datový typ `wztemporary`:

Příklad 1

```
VAR wztemporary roll;
...
PROC
  VAR shapedata volume;
  CONST pos t_center := [1000, 1000, 1000];
  ...
  WZCylDef \Inside, volume, t_center, 400, 1000;
  WZLimSup \Temp, roll, volume;
ENDPROC
```

Proměnná `wztemporary`, `roll` je definována jako válec. Jestliže robot dosáhne tohoto objemu, pohyb je zastaven.

Omezení

Data `wztemporary` mohou být definována jako proměnná (VAR) nebo jako perzistent (PERS). Mohou být globální v úloze nebo lokální v modulu, ale nikoliv lokální v rutině.

Argumenty typu `wztemporary` by měly být pouze celými daty, nikoliv elementem pole nebo komponentem záznamu.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Dočasná světová zóna musí být pouze definována (`WZLimSup` nebo `WZDOSet`) a uvolněna (`WZFree`) v pohybové úloze. Definice dočasných světových zón v jakémkoliv pozadí nejsou povoleny, protože mohou ovlivnit vykonávání programu v připojené pohybové úloze. Instrukce `WZDisable` a `WZEnable` se mohou používat v úloze v pozadí. Jestliže se vyskytne potřeba použít perzistentní proměnnou v multitaskingovém systému, nastavte init hodnotu na 0 v obou úlohách, např. PERS wztemporary share_workarea := [0];

Další příklady

Kompletní příklad můžete vidět v instrukci `WZDOSet`.

Konstrukce

```
< dataobject of wztemporary >
  < wz of num >
```

Související informace

Pro informace o	Viz
Světové zóny	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Světové zóny*
Tvar světové zóny	*shapedata - Data tvaru světové zóny na str 1578*
Stacionární světová zóna	*wzstationary - Data stacionární světové zóny na str 1638*
Aktivovat dohled limitu světové zóny	*WZLimSup - Aktivovat dohled limitu světové zóny na str 1022*
Aktivovat nastavení digitálního výstupu světové zóny	*WZDOSet - Aktivovat světovou zónu pro nastavení digitálního výstupu na str 1007*
Deaktivovat světovou zónu	*WZDisable - Deaktivovat dohled dočasné světové zóny na str 1005*
Aktivovat světovou zónu	*WZEnable - Aktivovat dohled dočasné světové zóny na str 1011*
Vymazat světovou zónu	*WZFree - Vymazat dohled dočasné světové zóny na str 1013*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.101 zonedata - Zónová data

Použití

`zonedata` se používá k určení, jak bude pozice ukončena, tj. jak blízko k naprogramované pozici musí být osy před pohybem dopředu k další pozici.

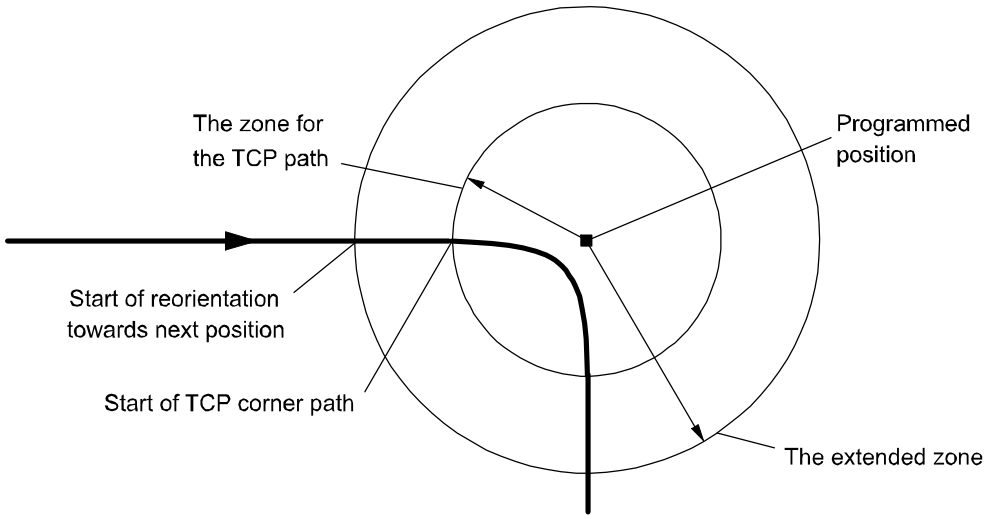
Popis

Pozice může být ukončena buď formou stop bodu nebo průjezdného bodu.

Stop bod znamená, že robot a externí osy musí dosáhnout určené pozice (stojí v klidu) předtím, než vykonávání programu pokračuje s další instrukcí. Je také možné definovat stop body jiné než předdefinované `fine`. Stop kritérium, které říká, jestli se má za to, že robot dosáhl bodu, může být manipulováno pomocí `stoppointdata`.

Průjezdný bod znamená, že naprogramované pozice není nikdy dosaženo. Namísto toho je směr pohybu změněn před dosažením pozice. Dvě odlišné zóny (rozsahy) mohou být definovány pro každou pozici:

- Zóna pro dráhu TCP.
- Rozšířená zóna pro reorientaci nástroje a pro externí osy.



xx0500002357

Funkce zón je stejná během společného pohybu, ale velikost zóny se může poněkud lišit od naprogramované.

Velikost zóny nemůže být větší než je polovina vzdálenosti k nejbližší pozici (dopředu nebo dozadu). Jestliže je určena větší zóna, robot ji automaticky zmenší.

Zóna pro dráhu TCP

Rohová dráha (parabola) je generována, jakmile je dosaženo okraje zóny (viz obrázek nahoře).

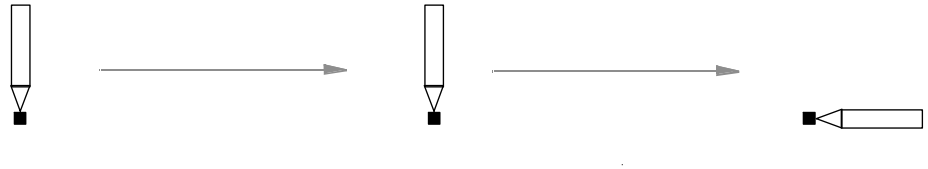
Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Zóna pro reorientaci nástroje

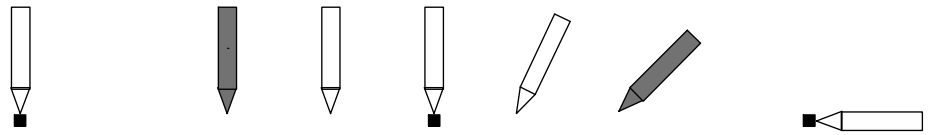
Reorientace začíná, jakmile TCP dosáhne rozšířené zóny. Nástroj je reorientován tak, že orientace je stejná při opouštění zóny, jaká by byla ve stejné pozici, jestliže stop body byly naprogramovány. Reorientace bude jemnější, jestliže velikost zóny je zvětšena, a existuje menší riziko nutnosti snížit rychlost pro provedení reorientace.

Následující obrázek ukazuje tři naprogramované pozice, poslední s odlišnou orientací nástroje.



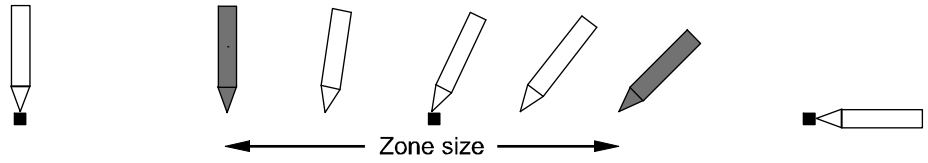
xx0500002358

Následující obrázek ukazuje, jak bude vypadat vykonávání programu, jestliže všechny pozice budou stop body.



xx0500002359

Následující obrázek ukazuje, jak bude vypadat vykonávání programu, jestliže střední pozice bude průjezdným bodem.



xx0500002360

Zóna pro externí osy

Externí osy se začínají pohybovat k další pozici, jakmile TCP dosáhne rozšířené zóny. Tímto způsobem může pomalá osa začít zrychlovat v rané fázi a tedy provádět jemněji.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.101 zonedata - Zónová data

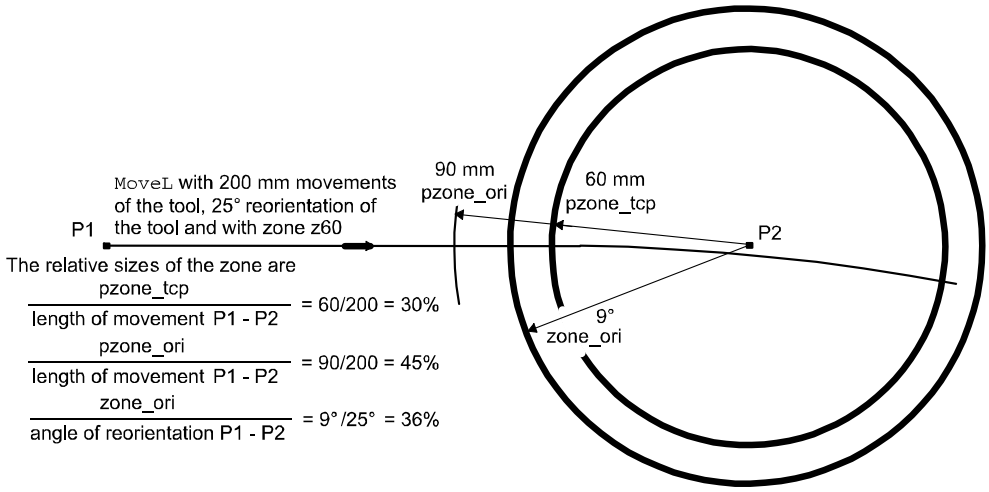
RobotWare - OS

Pokračování

Zmenšená zóna

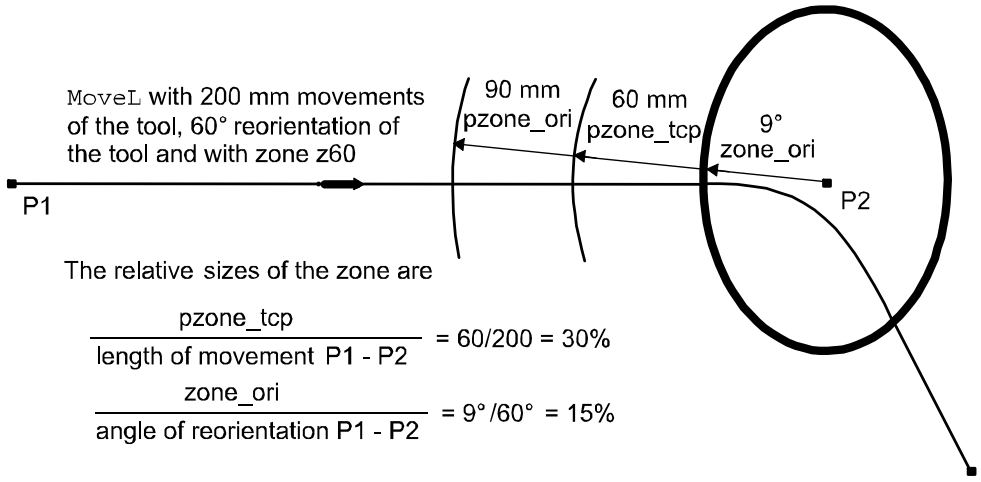
Při velkých reorientacích nástroje nebo při velkých pohybech externích os může být rozšířená zóna a dokonce TCP zóna zmenšena robotem. Zóna bude definována jako nejmenší relativní velikost zóny na základě komponentů zóny (viz *Komponenty na str 1645*) a naprogramovaného pohybu.

Následující obrázek ukazuje příklad zmenšené zóny pro reorientaci nástroje na 36 % pohybu kvůli zone_ori.



xx0500002362

Následující obrázek ukazuje příklad zmenšené zóny pro reorientaci nástroje a dráhu TCP na 15 % pohybu kvůli zone_ori.



xx0500002363

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Když jsou externí osy aktivní, ovlivňují relativní velikost zóny podle těchto vzorců:

$$\frac{\text{pzone\_eax}}{\text{length of movement P1 - P2}}$$

$$\frac{\text{zone\_leax}}{\text{length of max linear ext. axis movement P1 - P2}}$$

$$\frac{\text{zone\_reax}}{\text{angle of max reorientation of rotating ext. axis P1 - P2}}$$

xx0500002364

POZNÁMKA

Jestliže TCP zóna je zmenšena kvůli `zone_ori`, `zone_leax` nebo `zone_reax`, plánovač dráhy vstupuje do režimu, který může vyřešit případ žádného pohybu TCP. Jestliže existuje pohyb TCP v tomto režimu, rychlost není kompenzována vůči zakřivení dráhy v rohové zóně. Například, toto způsobí snížení rychlosti o 30 % v 90stupňovém rohu. Jestliže toto je problém, zvyšte omezovací komponent zóny.

Komponenty

`finep`

fine point

Datový typ: `bool`

Definuje, jestli pohyb bude ukončen jako stop bod (`fine` bod) nebo jako průjezdný bod.

- `TRUE`: **Pohyb končí jako stop bod a vykonávání programu nebude pokračovat, dokud robot nedosáhne stop bodu. Zbývající komponenty v zónových datech se nepoužívají.**
- `FALSE`: **Pohyb končí jako průjezdný bod a vykonávání programu pokračuje asi 100 ms předtím, než robot dosáhne zóny.**

`pzone_tcp`

path zone TCP

Datový typ: `num`

Velikost (poloměr) TCP zóny v mm.

Rozšířená zóna bude definována jako nejmenší relativní velikost zóny na základě následujících komponentů `pzone_ori...zone_reax` a naprogramovaného pohybu.

`pzone_ori`

path zone orientation

Datový typ: `num`

Velikost zóny (poloměr) pro reorientaci nástroje. Velikost je definována jako vzdálenost TCP od naprogramovaného bodu v mm.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

3.101 zonedata - Zónová data
RobotWare - OS
Pokračování

Velikost musí být větší než je odpovídající hodnota pro `pzone_tcp`. Jestliže je určena nižší hodnota, velikost je automaticky zvětšena, aby byla stejná jako `pzone_tcp`.

`pzone_eax`

path zone external axes

Datový typ: `num`

Velikost zóny (poloměr) pro externí osy. Velikost je definována jako vzdálenost TCP od naprogramovaného bodu v mm.

Velikost musí být větší než je odpovídající hodnota pro `pzone_tcp`. Jestliže je určena nižší hodnota, velikost je automaticky zvětšena, aby byla stejná jako `pzone_tcp`.

`zone_ori`

zone orientation

Datový typ: `num`

Velikost zóny pro reorientaci nástroje ve stupních. Jestliže robot drží pracovní objekt, znamená to úhel rotace pro pracovní objekt.

`zone_leax`

zone linear external axes

Datový typ: `num`

Velikost zóny pro lineární externí osy v mm.

`zone_reax`

zone rotational external axes

Datový typ: `num`

Velikost zóny pro rotační externí osy ve stupních.

Základní příklady

Následující příklad názorně ukazuje datový typ `zonedata`:

Příklad 1

```
VAR zonedata path := [ FALSE, 25, 40, 40, 10, 35, 5 ];
```

Zónová data `path` jsou definována prostřednictvím následujících vlastností:

- Velikost zóny pro dráhu TCP je `25` mm.
- Velikost zóny pro reorientaci nástroje je `40` mm (pohyb TCP).
- Velikost zóny pro externí osy je `40` mm (pohyb TCP).

Jestliže TCP stojí v klidu nebo je zde velká reorientace nebo velký pohyb externí osy s ohledem na zónu, platí místo toho následující:

- Velikost zóny pro reorientaci nástroje je `10` stupňů.
- Velikost zóny pro lineární externí osy je `35` mm.
- Velikost zóny pro rotační externí osy je `5` stupňů.

```
path.pzone_tcp := 40;
```

Velikost zóny pro dráhu TCP je nastavena na `40` mm.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Předdefinovaná data

Řada zónových dat je již definována v systému.

Stop body

Použijte `zonedata` pojmenovaná `fine`.

Průjezdové body

Zóny dráhy				Zóna		
Název	**Dráha TCP**	**Orientace**	**Ext.osa**	**Orientace**	**Lineární osa**	**Rotační osa**
z0	0,3 mm	0,3 mm	0,3 mm	0,03°	0,3 mm	0,03°
z1	1 mm	1 mm	1 mm	0,1°	1 mm	0,1°
z5	5 mm	8 mm	8 mm	0,8°	8 mm	0,8°
z10	10 mm	15 mm	15 mm	1,5°	15 mm	1,5°
z15	15 mm	23 mm	23 mm	2,3°	23 mm	2,3°
z20	20 mm	30 mm	30 mm	3,0°	30 mm	3,0°
z30	30 mm	45 mm	45 mm	4,5°	45 mm	4,5°
z40	40 mm	60 mm	60 mm	6,0°	60 mm	6,0°
z50	50 mm	75 mm	75 mm	7,5°	75 mm	7,5°
z60	60 mm	90 mm	90 mm	9,0°	90 mm	9,0°
z80	80 mm	120 mm	120 mm	12°	120 mm	12°
z100	100 mm	150 mm	150 mm	15°	150 mm	15°
z150	150 mm	225 mm	225 mm	23°	225 mm	23°
z200	200 mm	300 mm	300 mm	30°	300 mm	30°

Konstrukce

```
< data object of zonedata >
  < finep of bool >
  < pzone_tcp of num >
  < pzone_ori of num >
  < pzone_eax of num >
  < zone_ori of num >
  < zone_leax of num >
  < zone_reax of num >
```

Související informace

Pro informace o	Viz
Polohovací instrukce	*Technická referenční příručka - Přehled RAPID*, sekce *Souhrn RAPID - Pohyb*
Pohyby/Dráhy všeobecně	*Technická referenční příručka - Přehled RAPID*, sekce*Principy pohybu a I/O - Polohování během vykonávání programu*
Konfigurace externích os	*Application manual - Additional axes and stand alone controller*
Ostatní stop body	*stoppointdata - Data stop bodu na str 1590*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Tato stránka je záměrně prázdná

4 Příklady typu programování

4.1 Chybový handler (ERROR handler) s pohyby

Použití

Tyto typové příklady popisují, jak používat pohybové instrukce v chybovém handleru poté, co se objevil asynchronně vyvolaný proces nebo pohybová chyba.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion.

Popis

Chybový handler `ERROR` může spustit nový dočasný pohyb a nakonec restartovat původní přerušený a zastavený pohyb. Například, může se použít k přechodu do servisní pozice nebo k vyčištění pistole poté, co se objevil asynchronně vyvolaný proces nebo pohybová chyba.

Aby bylo možné dosáhnout této funkčnosti, musí se použít instrukce `StorePath - RestoPath` v chybovém handleru `ERROR`. Pro restart pohybu a pokračování ve vykonávání programu je k dispozici několik instrukcí `RAPID`.

Typové příklady

Typové příklady funkčnosti jsou názorně uvedeny dole.

Princip

```
...
ERROR
  IF ERRNO = ERR_PATH_STOP THEN
    StorePath;
    ! Move away and back to the interrupted position
    ...
    RestoPath;
    StartMoveRetry;
  ENDIF
ENDPROC
```

Při vykonávání `StartMoveRetry` robot obnoví svůj pohyb, všechny aktivní procesy jsou restartovány a program se pokusí o obnovu svého vykonávání. `StartMoveRetry` provádí to samé jako `StartMove` plus `RETRY` v jedné nedělitelné operaci.

Automatický restart vykonávání

```
CONST robtarget service_pos := [...];
VAR robtarget stop_pos;
...
ERROR
  IF ERRNO = AW_WELD_ERR THEN
    ! Current movement on motion base path level
    ! is already stopped.
    ! New motion path level for new movements in the ERROR handler
    StorePath;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

4.1 Chybový handler (ERROR handler) s pohyby

Path Recovery

Pokračování

```
    ! Store current position from motion base path level
    stop_pos := CRobT(\Tool:=tool1, \WObj:=wobj1);
    ! Do the work to fix the problem
    MoveJ service_pos, v50, fine, tool1, \WObj:=wobj1;
    ...
    ! Move back to the position on the motion base path level
    MoveJ stop_pos, v50, fine, tool1, \WObj:=wobj1;
    ! Go back to motion base path level
    RestoPath;
    ! Restart the stopped movements on motion base path level,
    ! restart the process and retry program execution
    StartMoveRetry;
  ENDIF
ENDPROC
```

Toto je typový příklad, jak používat automatickou asynchronní obnovu po chybě po určitém typu procesní chyby během pohybů robotu.

Ruční restart vykonávání

```
...
ERROR
  IF ERRNO = PROC_ERR_XXX THEN
    ! Current movement on motion base path level
    ! is already stopped and in stop move state.
    ! This error must be handle manually.
    ! Reset the stop move state on motion base path level.
    StopMoveReset;
  ENDIF
ENDPROC
```

Toto je typový příklad, jak používat ruční ošetření asynchronní obnovy po chybě po určitém typu procesní chyby během pohybů robotu.

Po vykonání shora uvedeného chybového handleru `ERROR` do konce se vykonávání programu zastaví a ukazatel programu je na začátku instrukce s procesní chybou (také na začátku každé použité rutiny NOSTEPIN). Další spuštění programu restartuje program a pohyb od pozice, ve které původní procesní chyba vznikla.

Vykonávání programu

Chování vykonávání:

- Na začátku vykonávání chybového handleru `ERROR` program opouští svoji základní úroveň vykonávání
- Při vykonávání `StorePath` pohybový systém opouští svoji základní úroveň vykonávání
- Při vykonávání `RestoPath` se pohybový systém vrací na svoji základní úroveň vykonávání
- Při vykonávání `StartMoveRetry` se program vrací na svoji základní úroveň vykonávání

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Omezení

Následující instrukce RAPID se musí používat v chybovém handleru ERROR s pohybovými instrukcemi, aby mohly fungovat při automatické obnově po chybě po asynchronně vyvolaném procesu nebo chybě dráhy:

Instrukce	Popis
StorePath	Vložte novou úroveň dráhy pohybu
RestoPath	Vraťte se k základní úrovni dráhy pohybu
StartMoveRetry	Restartujte přerušené pohyby na základní úrovni dráhy pohybu. Restartuje také proces a zkuste vykonávání programu. Stejná funkčnost jako StartMove + RETRY.

Následující instrukce RAPID se musí používat v chybovém handleru ERROR s pohybovými instrukcemi, aby mohly fungovat při ruční obnově po chybě po asynchronně vyvolaném procesu nebo chybě dráhy:

Instrukce	Popis
StopMoveReset	Vložte novou úroveň dráhy pohybu

Související informace

Pro informace o	Viz
Vložení nové úrovně dráhy pohybu	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742*
Návrat k základní úrovni dráhy pohybu	*RestoPath - Obnovuje cestu po přerušení na str 546*
Restart přerušeného pohybu, procesu, a pokus o vykonávání programu.	*StartMoveRetry - Restartuje pohyb robotu a vykonávání na str 710*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

4.2 Servisní rutiny s pohyby nebo bez pohybů

Použití

Tyto typové příklady popisují, jak používat pohybové instrukce v servisní rutině. Stejný princip u `StopMove`, `StartMove` a `StopMoveReset` je také platný u servisních rutin bez pohybů (pouze logické instrukce).

Jak servisní rutiny, tak i ostatní rutiny (procedury) bez parametrů mohou být spuštěny ručně a provádějí pohyby podle těchto typových příkladů.

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion v nezávislém nebo polokoordinovaném režimu.

Popis

Servisní rutina může spustit nový, dočasný pohyb a při pozdějším spuštění programu restartovat původní pohyb. Například, může se používat k přechodu na servisní pozici nebo ruční spuštění čištění pistole.

Aby bylo dosaženo této funkčnosti, musí se v servisní rutině použít instrukce `StorePath - RestoPath` a `StopMoveReset`.

Typové příklady

Typové příklady funkčnosti jsou názorně uvedeny dole.

Princip

```
PROC xxxx()
  StopMove;
  StorePath;
  ! Move away and back to the interrupted position
  ...
  RestoPath;
  StopMoveReset;
ENDPROC
```

`StopMove` je vyžadován, aby bylo zajištěno, že původní zastavený pohyb není restartován při ruční sekvenci „zastavit program-restartovat program“ během vykonávání servisní rutiny.

Stop na dráze

```
VAR robtarget service_pos := [...];
...
PROC proc_stop_on_path()
  VAR robtarget stop_pos;
  ! Current stopped movements on motion base path level
  ! must not be restarted in the service routine.
  StopMove;
  ! New motion path level for new movements in the service routine.
  StorePath;
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=tool1 \WObj:=wobj1);
  ! Do the work
  MoveJ service_pos, v50, fine, tool1 \WObj:=wobj1;
  ...
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, tool1, \WObj:=wobj1;
  ! Go back to motion base path level
  RestoPath;
  ! Reset the stop move state for the interrupted movement
  ! on motion base path level
  StopMoveReset;
ENDPROC
```

V tomto typovém příkladu pohyby v servisní rutině začínají a končí na pozici na dráze, kde byl program zastaven.

Všimněte si také, že použitý nástroj a pracovní objekt jsou známy v době programování.

Zastavit v dalším stop bodu

```
TASK PERS tooldata used_tool := [...];
TASK PERS wobjdata used_wobj := [...];
...
PROC proc_stop_in_stop_point()
  VAR robtarget stop_pos;
  ! Current move instruction on motion base path level continue to
       its ToPoint and will be finished in a stop point.
  StartMove;
  ! New motion path level for new movements in the service routine
  StorePath;
  ! Get current tool and work object data
  GetSysData used_tool;
  GetSysData used_wobj;
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=used_tool \WObj:=used_wobj);
  ! Do the work
  MoveJ Offs(stop_pos,0,0,20),v50,fine,used_tool\WObj:=used_wobj;
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, used_tool,\WObj:=used_wobj;
  ! Go back to motion base path level
  RestoPath;
  ! Reset the stop move state for any new movement
  ! on motion base path level
  StopMoveReset;
ENDPROC
```

V tomto typovém příkladu pohyby v servisní rutině pokračují a končí na `ToPoint` v instrukcích přerušeného pohybu, předtím, než je připravena instrukce `StorePath`.

Všimněte si také, že použitý nástroj a pracovní objekt jsou neznámé v době programování.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

4.2 Servisní rutiny s pohyby nebo bez pohybů

Path recovery

Pokračování

Vykonávání programu

Chování vykonávání:

- Na začátku vykonávání servisní rutiny program opouští svoji základní úroveň vykonávání
- Při vykonávání `StorePath` pohybový systém opouští svoji základní úroveň vykonávání
- Při vykonávání `RestoPath` se pohybový systém vrací na svoji základní úroveň vykonávání
- Při vykonávání `ENDPROC` se program vrací na svoji základní úroveň vykonávání

Omezení

Následující instrukce `RAPID` se musí používat v servisní rutině s pohybovými instrukcemi, aby bylo zajištěno fungování:

Instrukce	Popis
`StorePath`	Vložte novou úroveň dráhy pohybu
`RestoPath`	Vraťte se k základní úrovni dráhy pohybu
`StopMoveReset`	Resetujte stav zastavení pohybu pro přerušený pohyb na základní úrovni dráhy pohybu

Související informace

Pro informace o	Viz
Žádný restart již zastaveného pohybu na základní úrovni dráhy pohybu	*StopMove - Zastavuje pohyby robotu na str 736*
Restartovat již zastavený pohyb na základní úrovni dráhy pohybu	*StopMove - Zastavuje pohyby robotu na str 736*
Vložení nové úrovně dráhy pohybu	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742*
Návrat k základní úrovni dráhy pohybu	*RestoPath - Obnovuje cestu po přerušení na str 546*
Resetujte stav zastavení pohybu pro přerušený pohyb na základní úrovni dráhy pohybu	*StopMoveReset - Resetovat pohybový stav zastavení systému na str 740*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

4.3 Systémová přerušení I/O s pohyby nebo bez pohybů

Použití

Tyto typové příklady popisují, jak používat pohybové instrukce v rutině přerušení systémového I/O. Stejný princip u `StopMove`, `StartMove` a `StopMoveReset` je také platný u přerušení systémových I/O bez pohybů (pouze logické instrukce).

Tuto instrukci je možné používat pouze v hlavní úloze `T_ROB1` nebo v systému MultiMove v úlohách Motion v nezávislém nebo polokoordinovaném režimu.

Popis

Rutina přerušení systémového I/O může spustit nový, dočasný pohyb a při pozdějším spuštění programu restartovat původní pohyb. Například, může se používat k přechodu na servisní pozici nebo k čištění pistole, když se objeví přerušení.

Aby bylo dosaženo této funkčnosti, musí se v rutině přerušení servisních I/O použít instrukce `StorePath - RestoPath` a `StopMoveReset`.

Typové příklady

Typové příklady funkčnosti jsou názorně uvedeny dole.

Princip

```
PROC xxxx()
  StopMove;
  StorePath;
  ! Move away and back to the interrupted position
  ...
  RestoPath;
  StopMoveReset;
ENDPROC
```

`StopMove` se vyžaduje kvůli zajištění, aby původně zastavený pohyb nebyl restartován na začátku rutiny přerušení I/O.

Bez `StopMove` nebo s `StartMove` namísto toho pohyb v rutině přerušení I/O bude pokračovat okamžitě a končí u `ToPoint` v instrukci přerušeného pohybu.

Stop na dráze

```
VAR robtarget service_pos := [...];
...
PROC proc_stop_on_path()
  VAR robtarget stop_pos;
  ! Current stopped movements on motion base path level is nnt
       restarted in the system I/O routine.
  StopMove \Quick;
  ! New motion path level for new movements in the system
  ! I/O routine.
  StorePath;
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=tool1 \WObj:=wobj1);
  ! Do the work
  MoveJ service_pos, v50, fine, tool1 \WObj:=wobj1;
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

4.3 Systémová přerušení I/O s pohyby nebo bez pohybů
Path recovery
Pokračování

```
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, tool1, \WObj:=wobj1;
  ! Go back to motion base path level
  RestoPath;
  ! Reset the stop move state for the interrupted movement
  ! on motion base path level
  StopMoveReset;
ENDPROC
```

V tomto typovém příkladu jsou přerušené pohyby zastaveny náhle a jsou restartovány při spuštění programu po dokončení rutiny přerušení systémového I/O.

Všimněte si také, že použitý nástroj a pracovní objekt jsou známy v době programování.

Zastavit v dalším stop bodu

```
TASK PERS tooldata used_tool := [...];
TASK PERS wobjdata used_wobj := [...];
...
PROC proc_stop_in_stop_point()
  VAR robtarget stop_pos;
  ! Current move instruction on motion base path level continue to
       its ToPoint and will be finished in a stop point.
  StartMove;
  ! New motion path level for new movements in the system
  ! I/O routine
  StorePath;
  ! Get current tool and work object data
  GetSysData used_tool;
  GetSysData used_wobj;
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=used_tool \WObj:=used_wobj);
  ! Do the work
  MoveJ Offs(stop_pos,0,0,20),v50,fine,used_tool\WObj:=used_wobj;
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, used_tool,\WObj:=used_wobj;
  ! Go back to motion base path level
  RestoPath;
  ! Reset the stop move state for new movement
  ! on motion base path level
  StopMoveReset;
ENDPROC
```

V tomto typovém příkladu pohyby v rutině přerušení systémového I/O pokračují okamžitě a končí u `ToPoint` v instrukcích přerušeného pohybu.

Všimněte si také, že použitý nástroj a pracovní objekt jsou neznámé v době programování.

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Vykonávání programu

Chování vykonávání:

- Na začátku vykonávání rutiny systémového I/O program opouští svoji základní úroveň vykonávání
- Při vykonávání `StorePath` pohybový systém opouští svoji základní úroveň vykonávání
- Při vykonávání `RestoPath` se pohybový systém vrací na svoji základní úroveň vykonávání
- Při vykonávání `ENDPROC` se program vrací na svoji základní úroveň vykonávání

Omezení

Následující instrukce `RAPID` se musí používat v rutině systémového I/O s pohybovými instrukcemi, aby bylo zajištěno fungování:

Instrukce	Popis
`StorePath`	Vložte novou úroveň dráhy pohybu
`RestoPath`	Vraťte se k základní úrovni dráhy pohybu
`StopMoveReset`	Resetujte stav zastavení pohybu pro přerušený pohyb na základní úrovni dráhy pohybu

Související informace

Pro informace o	Viz
Žádný restart již zastaveného pohybu na základní úrovni dráhy pohybu	*StopMove - Zastavuje pohyby robotu na str 736*
Restartovat již zastavený pohyb na základní úrovni dráhy pohybu	*StartMove - Znovu spouští pohyb robotu na str 707*
Vložení nové úrovně dráhy pohybu	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742*
Návrat k základní úrovni dráhy pohybu	*RestoPath - Obnovuje cestu po přerušení na str 546*
Resetujte stav zastavení pohybu pro přerušený pohyb na základní úrovni dráhy pohybu	*StopMoveReset - Resetovat pohybový stav zastavení systému na str 740*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

4.4 TRAP rutiny s pohyby

Použití

Tyto typové příklady popisují, jak používat pohybové instrukce v rutině TRAP po vzniku přerušení.

Tuto funkčnost je možné používat pouze v hlavní úloze T_ROB1 nebo v systému MultiMove v úlohách Motion.

Popis

TRAPrutina může spustit nový, dočasný pohyb a nakonec restartovat původní pohyb. Například, může se používat k přechodu na servisní pozici nebo k čištění pistole, když se objeví přerušení.

Aby bylo dosaženo této funkčnosti, musí se použít StorePath - RestoPath a StartMove v TRAP rutině.

Typové příklady

Typové příklady funkčnosti jsou názorně uvedeny dole.

Princip

```
TRAP xxxx
  StopMove;
  StorePath;
  ! Move away and back to the interrupted position
  ...
  RestoPath;
  StartMove;
ENDTRAP
```

Jestliže je použit StopMove, pohyb se zastaví náhle na probíhající dráze; jinak pohyb pokračuje k ToPoint v aktuální pohybové instrukci.

Zastavit v dalším stop bodu

```
VAR robtarget service_pos := [...];
...
TRAP trap_in_stop_point
  VAR robtarget stop_pos;
  ! Current move instruction on motion base path level continue
  ! to it's ToPoint and will be finished in a stop point.
  ! New motion path level for new movements in the TRAP
  StorePath;
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=tool1 \WObj:=wobj1);
  ! Do the work
  MoveJ service_pos, v50, fine, tool1 \WObj:=wobj1;
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, tool1, \WObj:=wobj1;
  ! Go back to motion base path level
  RestoPath;
  ! Restart the interupted movements on motion base path level
```

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

```
  StartMove;
ENDTRAP
```

V tomto typovém příkladu pohyby v TRAP rutině začínají a končí u `ToPoint` v instrukcích přerušeného pohybu. Všimněte si také, že nástroj a pracovní objekt jsou známy v době programování.

Náhlé zastavení na dráze

```
TASK PERS tooldata used_tool := [...];
TASK PERS wobjdata used_wobj := [...];
...
TRAP trap_stop_at_once
  VAR robtarget stop_pos;
  ! Current move instruction on motion base path level stops
  ! at once
  StopMove;
  ! New motion path level for new movements in the TRAP
  StorePath;
  ! Get current tool and work object data
  GetSysData used_tool;
  GetSysData used_wobj;
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=used_tool \WObj:=used_wobj);
  ! Do the work
  MoveJ Offs(stop_pos,0,0,20),v50,fine,used_tool\WObj:=used_wobj;
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, used_tool,\WObj:=used_wobj;
  ! Go back to motion base path level
  RestoPath;
  ! Restart the interupted movements on motion base path level
  StartMove;
ENDTRAP
```

V tomto typovém příkladu pohyby v TRAP rutině začínají a končí na pozici na dráze, kde byla zastavena instrukce přerušeného pohybu. Všimněte si také, že nástroj a pracovní objekt jsou neznámé v době programování.

Vykonávání programu

Chování vykonávání:

- Na začátku vykonávání rutiny `TRAP` program opouští svoji základní úroveň vykonávání
- Při vykonávání `StorePath` pohybový systém opouští svoji základní úroveň vykonávání
- Při vykonávání `RestoPath` se pohybový systém vrací na svoji základní úroveň vykonávání
- Při vykonávání `ENDTRAP` se program vrací na svoji základní úroveň vykonávání

Pokračování na další straně

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

4.4 TRAP rutiny s pohyby

Path Recovery

Pokračování

Omezení

Následující instrukce `RAPID` se musí používat v `TRAP` rutině s pohybovými instrukcemi, aby bylo zajištěno fungování:

Instrukce	Popis
`StorePath`	Vložte novou úroveň dráhy pohybu
`RestoPath`	Vraťte se k základní úrovni dráhy pohybu
`StartMove`	Restartovat přerušené pohyby na základní úrovni dráhy pohybu

Související informace

Pro informace o	Viz
Náhlé zastavení aktuálního pohybu	*StopMove - Zastavuje pohyby robotu na str 736*
Vložení nové úrovně dráhy pohybu	*StorePath - Uloží dráhu, kde se přerušení objeví na str 742*
Návrat k základní úrovni dráhy pohybu	*RestoPath - Obnovuje cestu po přerušení na str 546*
Restart přerušeného pohybu	*StartMove - Znovu spouští pohyb robotu na str 707*

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Rejstřík

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

X

Z

© Copyright 2004-2016 ABB. Všechna práva vyhrazena.

Contact us

ABB AB
Discrete Automation and Motion
Robotics
S-721 68 VÄSTERÅS, Sweden
Telephone +46 (0) 21 344 400

ABB AS, Robotics
Discrete Automation and Motion
Nordlysvegen 7, N-4340 BRYNE, Norway
Box 265, N-4349 BRYNE, Norway
Telephone: +47 51489000

ABB Engineering (Shanghai) Ltd.
No. 4528 Kangxin Hingway
PuDong District
SHANGHAI 201319, China
Telephone: +86 21 6105 6666

www.abb.com/robotics

3HAC050917-014, Rev C, cs

Power and productivity
for a better world™