

应用手册 控制器软件IRC5

Power and productivity
for a better world™



Trace back information:
Workspace R16-1 version a6
Checked in 2016-03-01
Skribenta version 4.6.209

应用手册

控制器软件IRC5

RobotWare 6.03

文档编号: 3HAC050798-010

修订: C

本手册中包含的信息如有变更，恕不另行通知，且不应视为 ABB 的承诺。ABB 对本手册中可能出现的错误概不负责。

除本手册中有明确陈述之外，本手册中的任何内容不应解释为 ABB 对个人损失、财产损失或具体适用性等做出的任何担保或保证。

ABB 对因使用本手册及其中所述产品而引起的意外或间接伤害概不负责。

未经 ABB 的书面许可，不得再生或复制本手册和其中的任何部件。

可从 ABB 处获取此手册的额外复印件。

本出版物的原始语言为英语。所有其他语言版本均翻译自英语版本。

© 版权所有 2016 ABB。保留所有权利。

ABB AB
Robotics Products
Se-721 68 Västerås
瑞典

目表

手册概述	13
1 RobotWare介绍	15
2 RobotWare-OS	17
2.1 Advanced RAPID	17
2.1.1 Advanced RAPID介绍	17
2.1.2 位功能	18
2.1.2.1 概述	18
2.1.2.2 RAPID组件	19
2.1.2.3 位功能的示例	20
2.1.3 数据搜索功能	21
2.1.3.1 概述	21
2.1.3.2 RAPID组件	22
2.1.3.3 数据搜索功能的示例	23
2.1.4 别名 I/O 信号	24
2.1.4.1 概述	24
2.1.4.2 RAPID组件	25
2.1.4.3 别名 I / O 功能的示例	26
2.1.5 配置功能	27
2.1.5.1 概述	27
2.1.5.2 RAPID组件	28
2.1.5.3 配置功能的示例	29
2.1.6 断电功能	30
2.1.6.1 概述	30
2.1.6.2 RAPID组件与系统参数	31
2.1.6.3 断电功能的示例	32
2.1.7 工艺配套功能	33
2.1.7.1 概述	33
2.1.7.2 RAPID组件	34
2.1.7.3 工艺配套功能的示例	35
2.1.8 中断功能	37
2.1.8.1 概述	37
2.1.8.2 RAPID组件	38
2.1.8.3 中断功能的示例	39
2.1.9 用户消息功能	40
2.1.9.1 概述	40
2.1.9.2 RAPID组件	41
2.1.9.3 用户消息功能的示例	42
2.1.9.4 文本表格文件	43
2.1.10 RAPID配套功能	44
2.1.10.1 概述	44
2.1.10.2 RAPID组件	45
2.1.10.3 RAPID配套功能的示例	46
2.2 Analog Signal Interrupt	47
2.2.1 Analog Signal Interrupt介绍	47
2.2.2 RAPID组件	48
2.2.3 代码示例	49
2.3 Auto Acknowledge Input	50
2.4 Cyclic bool	51
2.4.1 循环评估逻辑条件	51
2.4.2 RAPID组件	53
2.4.3 Cyclic bool示例	54
2.5 Electronically Linked Motors	57
2.5.1 概述	57
2.5.2 配置	58
2.5.2.1 系统参数	58

2.5.2.2	配置示例	60
2.5.3	管理一根从动轴	61
2.5.3.1	使用服务程序	61
2.5.3.2	校准从动轴位置	62
2.5.3.3	重置从动轴	64
2.5.4	微调一个扭矩从动件	65
2.5.4.1	扭矩从动件的描述	65
2.5.4.2	使用服务程序	66
2.5.5	数据设置	68
2.5.5.1	设置服务程序的数据	68
2.5.5.2	数据设置示例	70
2.6	Fixed Position Events	72
2.6.1	概述	72
2.6.2	RAPID组件与系统参数	73
2.6.3	代码示例	75
2.7	File and Serial Channel Handling	77
2.7.1	File and Serial Channel Handling介绍	77
2.7.2	基于二进制和字符的通信	78
2.7.2.1	概述	78
2.7.2.2	RAPID组件	79
2.7.2.3	代码示例	80
2.7.3	原始数据通信	82
2.7.3.1	概述	82
2.7.3.2	RAPID组件	83
2.7.3.3	代码示例	84
2.7.4	文件与目录管理	86
2.7.4.1	概述	86
2.7.4.2	RAPID组件	87
2.7.4.3	代码示例	88
2.8	Device Command Interface	90
2.8.1	Device Command Interface介绍	90
2.8.2	RAPID部件和系统参数	91
2.8.3	代码示例	92
2.9	Logical Cross Connections	94
2.9.1	Logical Cross Connections介绍	94
2.9.2	配置Logical Cross Connections	95
2.9.3	示例	96
2.9.4	限制	98
2.10	Remote Service Embedded	99
2.10.1	概述	99
2.10.2	RSE连接	101
2.10.3	配置 - 系统参数	103
2.10.4	RSE注册	104
2.10.5	远程服务信息	106
3	Motion performance	111
3.1	Absolute Accuracy [603-1, 603-2]	111
3.1.1	关于Absolute Accuracy	111
3.1.2	何时使用Absolute Accuracy	112
3.1.3	有用工具	113
3.1.4	配置	114
3.1.5	维护	116
3.1.5.1	影响准确度的维护	116
3.1.5.2	丧失准确度	118
3.1.6	补偿理论	119
3.1.6.1	错误来源	119
3.1.6.2	Absolute Accuracy补偿	120
3.1.7	Absolute Accuracy机器人的准备	122
3.1.7.1	ABB校准进程	122

3.1.7.2	出厂证书	124
3.1.7.3	配置参数	125
3.1.8	围笼的对准	128
3.1.8.1	概述	128
3.1.8.2	测量固定装置的对准情况	129
3.1.8.3	测量机器人的对准情况	130
3.1.8.4	框架关系	131
3.1.8.5	工具校准	132
3.2	Advanced robot motion [687-1]	133
3.3	Advanced Shape Tuning 【包括在687-1中】	134
3.3.1	关于Advanced Shape Tuning	134
3.3.2	自动微调摩擦	135
3.3.3	手动微调摩擦	137
3.3.4	系统参数	138
3.3.4.1	系统参数	138
3.3.4.2	设置微调系统参数	139
3.3.5	RAPID组件	140
3.4	Motion Process Mode 【包括在687-1中】	141
3.4.1	关于Motion Process Mode	141
3.4.2	用户定义的模式	143
3.4.3	关于机器人微调的一般信息	145
3.4.4	附加信息	147
3.5	Wrist Move 【包括在687-1中】	148
3.5.1	Wrist Move介绍	148
3.5.2	切割面框架	149
3.5.3	RAPID组件	150
3.5.4	RAPID代码示例	151
3.5.5	故障排查	153
4	Motion coordination	155
4.1	Machine Synchronization [607-1], [607-2]	155
4.1.1	概述	155
4.1.2	所需事项	156
4.1.3	同步特性	158
4.1.4	对同步进程的一般性描述	159
4.1.5	限制	160
4.1.6	Sensor Synchronization的硬件安装	161
4.1.6.1	编码器规范	161
4.1.6.2	对编码器的描述	162
4.1.6.3	安装建议	163
4.1.6.4	连接编码器和编码器接口单元	164
4.1.7	Analog Synchronization的硬件安装	166
4.1.7.1	所需硬件	166
4.1.8	软件安装	167
4.1.8.1	传感器的安装	167
4.1.8.2	重新载入保存的Motion参数	169
4.1.8.3	多个传感器的安装	170
4.1.9	同步编程	171
4.1.9.1	用同步选项编程时的一般问题	171
4.1.9.2	编程示例	173
4.1.9.3	进入和退出角区中的协调运动	175
4.1.9.4	使用若干传感器	176
4.1.9.5	精确点编程	177
4.1.9.6	丢弃传感器对象	178
4.1.9.7	FlexPendant示教器上的信息	179
4.1.9.8	编写考虑因素	180
4.1.9.9	运行模式	182
4.1.10	机器人与机器人之间的同步	183
4.1.10.1	简介	183

4.1.10.2	“机器人与机器人之间的同步”概念	184
4.1.10.3	主动机器人的配置参数	185
4.1.10.4	从动机器人的配置参数	188
4.1.10.5	主动机器人的编程示例	191
4.1.10.6	从动机器人的编程示例	193
4.1.11	用已记录的曲线来与液压式冲压机同步	194
4.1.11.1	介绍	194
4.1.11.2	系统参数的配置	195
4.1.11.3	程序示例	197
4.1.12	用已记录的曲线来与注塑机同步	198
4.1.12.1	介绍	198
4.1.12.2	系统参数的配置	199
4.1.12.3	程序示例	201
4.1.13	监控	202
4.1.14	系统参数	203
4.1.15	I/O 信号	206
4.1.16	RAPID组件	207
5	Motion Events	209
5.1	World Zones [608-1]	209
5.1.1	概述	209
5.1.2	RAPID组件	211
5.1.3	代码示例	213
6	Motion functions	215
6.1	Independent Axes [610-1]	215
6.1.1	概述	215
6.1.2	系统参数	216
6.1.3	RAPID组件	217
6.1.4	代码示例	218
6.2	Path Recovery [611-1]	220
6.2.1	概述	220
6.2.2	RAPID组件	221
6.2.3	保存当前路径	222
6.2.4	路径记录	228
6.3	Path Offset [612-1]	235
6.3.1	概述	235
6.3.2	RAPID组件	236
6.3.3	相关的RAPID功能	237
6.3.4	代码示例	238
7	Motion Supervision	239
7.1	Collision Detection [613-1]	239
7.1.1	概述	239
7.1.2	限制	240
7.1.3	碰撞时的情况	241
7.1.4	附加信息	243
7.1.5	配置和编写设施	244
7.1.5.1	系统参数	244
7.1.5.2	RAPID组件	245
7.1.5.3	Signals	246
7.1.6	何时使用Collision Detection	247
7.1.6.1	设置系统参数	247
7.1.6.2	FlexPendant示教器的调节监控	248
7.1.6.3	用RAPID程序调节监控	249
7.1.6.4	如何避免误触发	250

8	Communication	251
8.1	FTP Client [614-1]	251
8.1.1	FTP Client介绍	251
8.1.2	系统参数	253
8.1.3	示例	254
8.2	NFS Client [614-1]	255
8.2.1	NFS Client介绍	255
8.2.2	系统参数	257
8.2.3	示例	258
8.3	PC Interface [616-1]	259
8.3.1	PC Interface介绍	259
8.3.2	发送RAPID的变量	260
8.3.3	使用PC接口的ABB软件	262
8.4	Socket Messaging [616-1]	263
8.4.1	Socket Messaging介绍	263
8.4.2	套接字通信的示意图	264
8.4.3	关于套接字消息发送的技术实情	265
8.4.4	RAPID组件	266
8.4.5	代码示例	268
8.5	RAPID Message Queue【包括在616-1、623-1中】	270
8.5.1	RAPID Message Queue介绍	270
8.5.2	RAPID消息队列行为	271
8.5.3	系统参数	274
8.5.4	RAPID组件	275
8.5.5	代码示例	276
9	Engineering tools	281
9.1	Multitasking [623-1]	281
9.1.1	Multitasking介绍	281
9.1.2	系统参数	283
9.1.3	RAPID组件	284
9.1.4	任务配置	285
9.1.4.1	设置任务的调试策略	285
9.1.4.2	优先级	287
9.1.4.3	任务面板设定	288
9.1.4.4	选择用“启动”按钮启动哪项任务	289
9.1.5	各项任务间的通信	291
9.1.5.1	永久变量	291
9.1.5.2	等候其它任务	292
9.1.5.3	多项任务之间的同步	294
9.1.5.4	使用调度程序	296
9.1.6	其它编程问题	298
9.1.6.1	在各项任务之间共享资源	298
9.1.6.2	测试任务是否控制着机械单元	299
9.1.6.3	taskid	300
9.1.6.4	避免冗长环路	301
9.2	Sensor Interface [628-1]	302
9.2.1	Sensor Interface介绍	302
9.2.2	配置传感器	303
9.2.2.1	关于这些传感器	303
9.2.2.2	通过串行通道来配置传感器	304
9.2.2.3	通过以太网通道来配置传感器	305
9.2.3	RAPID	306
9.2.3.1	RAPID组件	306
9.2.4	示例	308
9.2.4.1	代码示例	308

9.3	Externally Guided Motion [689-1]	310
9.3.1	EGM介绍	310
9.3.1.1	概述	310
9.3.1.2	EGM Position Guidance介绍	312
9.3.1.3	EGM Path Correction介绍	313
9.3.2	使用EGM	314
9.3.2.1	基本方法	314
9.3.2.2	执行状态	315
9.3.2.3	输入数据	316
9.3.2.4	输出数据	319
9.3.2.5	配置	320
9.3.2.6	框架	321
9.3.3	EGM传感器协议	323
9.3.4	系统参数	327
9.3.5	RAPID部件	328
9.3.6	RAPID代码示例	330
9.3.6.1	使用带一件UdpUc装置的EGM Position Guidance	330
9.3.6.2	使用带输入项信号的EGM Position Guidance	332
9.3.6.3	使用协议类型不同的EGM Path Correction	336
9.3.7	UdpUc代码示例	339
9.4	Robot Reference Interface 【包括在689-1中】	340
9.4.1	Robot Reference Interface介绍	340
9.4.2	安装	341
9.4.2.1	连接通信电缆	341
9.4.2.2	先决条件	342
9.4.2.3	数据编配	343
9.4.2.4	受支持的数据类型	344
9.4.3	配置	345
9.4.3.1	接口配置	345
9.4.3.2	接口设定	346
9.4.3.3	装置说明	347
9.4.3.4	装置配置	349
9.4.4	配置示例	351
9.4.4.1	RAPID编程	351
9.4.4.2	配置示例	352
9.4.5	RAPID组件	357
10	Tool control options	359
10.1	Servo Tool Change [630-1]	359
10.1.1	概述	359
10.1.2	要求与限制	360
10.1.3	配置	362
10.1.4	连接中继器	363
10.1.5	无返回值工具更换程序	364
10.1.6	被禁止激活的点动伺服工具	365
10.2	Tool Control [1180-1]	366
10.2.1	概述	366
10.2.2	伺服工具的移动	367
10.2.3	焊枪头管理	368
10.2.4	监控	369
10.2.5	RAPID组件	370
10.2.6	系统参数	371
10.2.7	调试与服务	375
10.2.8	机械单元的校准	376
10.2.9	RAPID代码示例	377
10.3	I/O Controlled Axes [包含在1180-1中]	378
10.3.1	概述	378
10.3.2	轮廓错误	379
10.3.3	纠正位置	380

10.3.4 工具变换	381
10.3.5 安装	382
10.3.6 配置	383
10.3.7 系统参数	385
10.3.8 RAPID编程	387
索引	389

此页刻意留白

手册概述

关于本手册

本手册说明了何时使用及如何使用各种RobotWare的选项与函数。

手册用法

用户即可参考本手册来判断某个选项是否正好能解决某一问题，也可按本手册的说明来使用某个选项。本手册不含RAPID例程或类似程序的详细语法信息，具体请参见它们各自的参考手册。

本手册的阅读对象

本手册供机器人程序员使用。

操作前提

读者宜满足以下要求：

- 熟悉工业机器人及其术语。
- 熟悉RAPID编程语言。
- 熟悉系统参数和这些参数的配置方法。

参考信息

参考文档	文档编号
产品规格 - 控制器软件IRC5 IRC5 及主计算机 DSQC1000 和 RobotWare 6。	3HAC050945-010
产品规格 - 控制器IRC5 IRC5 及主计算机 DSQC1000.	3HAC047400-010
操作员手册 - RobotStudio	3HAC032104-010
操作员手册 - 带 FlexPendant 的 IRC5	3HAC050941-010
技术参考手册 - RAPID指令、函数和数据类型	3HAC050917-010
技术参考手册 - RAPID语言概览	3HAC050947-010
技术参考手册 - 系统参数	3HAC050948-010

修订版

版本号	描述
-	随 RobotWare 6.0 发布。 第一版
A	随 RobotWare 6.01 发布。 <ul style="list-style-type: none">• 新增Auto Acknowledge Input, 具体请参见第50页的Auto Acknowledge Input。• 修正了RAPID Message Queue的功能, 具体请参见第270页的RAPID Message Queue【包括在616-1、623-1中】。• 细微纠正。

下一页继续

版本号	描述
B	<p>随RobotWare 6.02一同发布。</p> <ul style="list-style-type: none"> 更新了模板文件的路径，具体请参见第339页的<i>UdpUc</i>代码示例和第375页的<i>调试与服务</i>。 更新了选项Sensor Interface [628-1]的TCP端口和协议，具体请参见第305页的<i>通过以太网通道来配置传感器</i>。 为功能EGM Path Correction添加了对应的RAPID指令，具体请参见第310页的<i>Externally Guided Motion [689-1]</i>。 本手册中记录了与母选项对应的捆绑选项。 更新了用于光学跟踪的LTAPP变量清单，具体请参见第306页的<i>常数</i>。
C	<p>随 RobotWare 6.03 发布。</p> <ul style="list-style-type: none"> 增加第51页的<i>Cyclic bool</i>功能。 增加第99页的<i>Remote Service Embedded</i>功能。 增加并更新了第141页的<i>Motion Process Mode</i>【包括在687-1中】选项的功能。 选项<i>Servo Tool Control [included in 635-6]</i>由选项第366页的<i>Tool Control [1180-1]</i>替换。 增加了选项第378页的<i>I/O Controlled Axes [包含在1180-1中]</i>。 细微纠正。

1 RobotWare介绍

软件产品

RobotWare 是 ABB Robotics 的系列软件产品。此产品旨在提高生产效率以及降低机器人的拥有成本和运行成本。ABB Robotics 多年专注于这些产品的开发，它们代表了从数以千计的机器人安装中汲取的知识和经验。

产品类别

在 RobotWare 系列中，有不同的产品类别：

产品类别	描述
RobotWare-OS	<p>这是机器人的操作系统。RobotWare-OS 为基础机器人编程和运行提供了所有必要的功能。这是机器人的固有部分，但也可以单独提供来进行升级。</p> <p>有关 RobotWare-OS 的说明，请参阅 产品规格 - 控制器IRC5。</p>
RobotWare 选件	<p>这些产品是在 RobotWare-OS 上运行的选件。它们是为需要动作控制、通信、系统工程或应用等附加功能的机器人用户准备的。</p> <p> 注意</p> <p>本手册并未介绍完所有的RobotWare选项，一些更全面的选项请参见各份单独手册。更多信息请参见产品规格 - 控制器软件IRC5。</p>
生产应用选件	<p>这些是点焊、弧焊和分配等的特定生产应用的扩展包。它们主要是为了提升生产成果和简化应用的安装与编程而设计的。</p> <p>各份单独手册介绍了相关工艺应用的所有选项。更多信息请参见产品规格 - 控制器软件IRC5。</p>
RobotWare Add-ins	<p>RobotWare Add-in 是自包含包，可扩展机器人系统的功能。</p> <p>ABB Robotics的部分软件产品是以Add-ins的形式发布的，比如导轨运动IRBT、定位器IRBP和独立控制器等。更多信息请参见产品规格 - 控制器软件IRC5。</p> <p>RobotWare Add-ins 的目的还包括让 ABB 外部的程序开发者能为 ABB 机器人系统创建选件，并将选件销售给他们的客户。有关创建 RobotWare Add-ins 的更多详情请联系您当地的 ABB Robotics 代表，您可以在 www.abb.com/contacts 找到相关信息。</p>

选项组

就IRC5而言，RobotWare的各选项已根据客户的利益编成若干组，以便更好地了解客户的选项值。不过所有选项都要单独购买。这些编组如下：

选项组	描述
Motion performance	该组选项可优化您机器人的性能。
Motion coordination	该组选项可让您的机器人与外接设备或其它机器人相互协调。
Motion Events	该组选项可监管机器人的位置。
Motion functions	该组选项可控制机器人的路径。
Motion Supervision	该组选项可监管机器人的移动。
Communication	该组选项可让机器人与其它设备相互通信（外接PC等）。
Engineering tools	该组选项供高级机器人集成人员使用。
Servo motor control	该组选项可通过机器人控制器来运行独立于机器人的外部电机。

下一页继续



注意

本手册并未介绍完所有的RobotWare选项，一些更全面的选项请参见各份单独手册。
更多信息请参见产品规格 - 控制器软件*IRC5*。

2 RobotWare-OS

2.1 Advanced RAPID

2.1.1 Advanced RAPID介绍

Advanced RAPID介绍

RobotWare的基本功能*Advanced RAPID*可供机器人程序员开发那些需要高级功能的应用程序。

Advanced RAPID包括了许多不同类型的功能，这些功能可分成下列各组：

功能组	描述
位功能	在一个字节上逐位运算。
数据搜索功能	搜索并获取 / 设置数据对象（如变量等）。
别名输入 / 输出 (I/O) 功能	为一个I / O信号指定一个可选的别名。
配置功能	获取 / 设置系统参数。
断电功能	在断电后恢复信号。
工艺配套功能	可用于创建工艺应用。
中断功能	包括了RobotWare基本功能中没有的其它中断功能。
用户消息功能	错误消息及其它文本。
RAPID配套功能	为程序员提供的其它支持。

2.1.2.1 概述

2.1.2 位功能

2.1.2.1 概述

目的

位功能的用途是在一个字节上实现运算，比如8数位的运算。其既可获取或设置单独一个位，也可在一个字节上进行逻辑运算。这种运算可用于处理串行通信或数字I/O信号组。

其中包括

位功能包括：

- 数据类型byte。
- 用于设置一个位值的指令：BitSet和BitClear。
- 用于获取一个位值的函数：BitCheck。
- 用于在一个字节上进行逻辑运算的函数：BitAnd、BitOr、BitXOr、BitNeg、BitLSh和BitRSh。

2.1.2.2 RAPID组件

数据类型

此处简述了位功能所用的每种数据类型。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各种数据类型。

数据类型	描述
byte	数据类型byte代表了0到255之间的一个十进制值。

指令：

此处简述了位功能所用的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
BitSet	BitSet用于将已定义字节数据中的某个指定位设置成1。
BitClear	BitClear用于清除已定义字节数据中的某个指定位（即设置成0）。

函数

此处简述了位功能所用的每则函数。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各则函数。

功能	描述
BitAnd	BitAnd用于在数据类型字节上执行一次逻辑逐位AND运算。
BitOr	BitOr用于在数据类型字节上执行一次逻辑逐位OR运算。
BitXOr	BitXOr（位异或）用于在数据类型字节上执行一次逻辑逐位XOR运算。
BitNeg	BitNeg用于在数据类型字节上执行一次逻辑逐位非运算（一的补数）。
BitLSh	BitLSh（位左移）用于在数据类型字节上执行一次逻辑逐位左移位运算。
BitRSh	BitRSh（位右移）用于在数据类型字节上执行一次逻辑逐位右移位运算。
BitCheck	BitCheck用于检查已定义字节数据中的某个指定位是否被设置成1。



提示

字节与字符串之间的换算函数——StrToByte和ByteToStr——并非相关选项的一部分，不过人们经常将它们与位功能一同使用。

2.1.2.3 位功能的示例

程序代码

```
CONST num parity_bit := 8;

!Set data1 to 00100110
VAR byte data1 := 38;

!Set data2 to 00100010
VAR byte data2 := 34;

VAR byte data3;

!Set data3 to 00100010
data3 := BitAnd(data1, data2);

!Set data3 to 00100110
data3 := BitOr(data1, data2);

!Set data3 to 00000100
data3 := BitXOr(data1, data2);

!Set data3 to 11011001
data3 := BitNeg(data1);

!Set data3 to 10011000
data3 := BitLSh(data1, 2);

!Set data3 to 00010011
data3 := BitRSh(data1, 1);

!Set data1 to 10100110
BitSet data1, parity_bit;

!Set data1 to 00100110
BitClear data1, parity_bit;

!If parity_bit is 0, set it to 1
IF BitCheck(data1, parity_bit) = FALSE THEN
  BitSet data1, parity_bit;
ENDIF
```

2.1.3 数据搜索功能

2.1.3.1 概述

目的

数据搜索功能的用途是搜索特定类型的数据对象，并获取 / 设置这些对象的数值。

以下是数据搜索功能的一些应用示例：

- 在某变量的名称仅存在于字符串中的情况下，将某个数值设为该变量。
- 列出特定类型的所有变量。
- 为具有类似名称的一组类似变量设置一个新值。

其中包括

数据搜索功能包括：

- 数据类型datapos。
- 用于搜索一组数据对象，并获取或设置它们的数值：SetDataSearch、GetDataVal、SetDataVal和SetAllDataVal。
- 用于遍历搜索结果的一项函数：GetNextSym。

2.1.3.2 RAPID组件

数据类型

此处简述了数据搜索功能所用的每种数据类型。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各种数据类型。

数据类型	描述
datapos	datapos是用函数GetNextSym检索出的某数据对象（内部系统数据）的封闭块。

指令：

此处简述了数据搜索功能所用的每条指令。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各条指令。

指令	描述
SetDataSearch	SetDataSearch与GetNextSym一起用于从相关系统中检索出数据对象。
GetDataVal	有了GetDataVal，用户便可从某字符串变量指定的数据对象中获取一个数值，或从GetNextSym检索出的数据对象中获取一个数值。
SetDataVal	有了SetDataVal，用户便可在某字符串变量指定的数据对象中设置一个数值，或在GetNextSym检索出的数据对象中设置一个数值。
SetAllDataVal	SetAllDataVal可为了其类型符合指定语法的所有数据对象设置一个新值。

函数

此处简述了数据搜索功能所用的每则函数。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各则函数。

功能	描述
GetNextSym	GetNextSym（获取下一个符号）与SetDataSearch一起用于从相关系统中检索出数据对象。

2.1.3.3 数据搜索功能的示例

设置未知变量

此例展示了当某变量名称未知且仅出现于字符串中时，该如何在编程时设置该变量的数值。

```
VAR string my_string;
VAR num my_number;
VAR num new_value:=10;
my_string := "my_number";
!Set value to 10 for variable specified by my_string
SetDataVal my_string,new_value;
```

重置变量范围

在此例中，所有以“my”打头的数字变量都被重置为0。

```
VAR string my_string:="my.*";
VAR num zeroval:=0;
SetAllDataVal "num"\Object:=my_string,zeroval;
```

列出 / 设置特定变量

在此例中，模块“mymod”中所有以“my”打头的数字变量都被列在了FlexPendant示教器上，然后被重置为0。

```
VAR datapos block;
VAR string name;
VAR num valuevar;
VAR num zeroval:=0;

!Search for all num variables starting with "my" in the module
"mymod"
SetDataSearch "num"\Object:="my.*"\InMod:="mymod";

!Loop through the search result
WHILE GetNextSym(name,block) DO
  !Read the value from each found variable
  GetDataVal name\Block:=block,valuevar;

  !Write name and value for each found variable
  TPWrite name+" = "\Num:=valuevar;

  !Set the value to 0 for each found variables
  SetDataVal name\Block:=block,zeroval;
ENDWHILE
```

2.1.4 别名 I/O 信号

2.1.4.1 概述

目的

别名I / O功能使程序员能为一个信号指定任何名称，并将该名称与一个配置好I / O信号联系起来。

在不同系统中重复使用RAPID程序时这一点会发挥作用：用户不用重写代码，而是利用新系统上既有的信号名称来把该程序所用的信号名称定义为一个别名。

其中包括

别名I / O功能包含了指令AliasIO。

2.1.4.2 RAPID组件

数据类型

没有针对别名I / O功能的RAPID数据类型。

指令：

此处简述了别名I / O功能所用的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
AliasIO	AliasIO的作用是用一个别名来定义任何类型的信号，或使用内置任务模块中的信号。其别名与一个配置好的I / O信号有关。 使用任何实际信号前都必须先运行指令AliasIO。

函数

没有针对别名I / O功能的RAPID函数。

2.1.4.3 别名I / O功能的示例

指定信号的别名

该示例展示了如何定义与“配置好的数字输出I / O信号config_do”有关的数字输出信号alias_do。

路径prog_start与“启动”事件有关。

这将确保在即使没有用上述名称配置信号的情况下，也能在RAPID代码中使用“alias_do”。

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

2.1.5 配置功能

2.1.5.1 概述

目的

配置功能可让程序员在运行时访问各个系统参数，并对其进行读取和编辑。可重启相关控制器来让新的参数值生效。

其中包括

配置功能中包括了下列指令：ReadCfgData、WriteCfgData和WarmStart。

2.1.5.2 RAPID组件

数据类型

没有针对配置功能的RAPID数据类型。

指令：

此处简述了配置功能所用的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
ReadCfgData	ReadCfgData的作用是读取一个已命名系统参数（配置数据）的一项属性。
WriteCfgData	WriteCfgData的作用是写入一个已命名系统参数（配置数据）的一项属性。
WarmStart	WarmStart的作用是在运行时重启控制器。 这能在用指令WriteCfgData更改系统参数后发挥作用。

函数

没有针对配置功能的RAPID函数。

2.1.5.3 配置功能的示例

配置系统参数

在此处的示例中，用户读取了rob1_1的系统参数`cal_offset`，将其增加了0.2毫米，然后又重新写入。重启相关控制器后此更改才会生效。

```
VAR num old_offset;  
VAR num new_offset;  
  
ReadCfgData "/MOC/MOTOR_CALIB/rob1_1", "cal_offset",old_offset;  
new_offset := old_offset + (0.2/1000);  
WriteCfgData "/MOC/MOTOR_CALIB/rob1_1", "cal_offset",new_offset;  
WarmStart;
```

2.1.6.1 概述

2.1.6 断电功能

2.1.6.1 概述

目的

如果上电失败时机器人正在路径上移动，那么机器人恢复运动时就可能需要一些额外的行动。断电功能可帮您检测路径移动期间是否发生了上电失败。



注意

更多信息请参见技术参考手册 - 系统参数中主题 *I/O System* 下的类型 *Signal Safe Level*。

其中包括

断电功能中包括了一个检查被中断路径的函数：`PFRestart`

2.1.6.2 RAPID组件与系统参数

数据类型

没有针对断电功能的RAPID数据类型。

指令：

没有针对断电功能的RAPID指令。

函数

此处简述了断电功能所用的每则函数。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各则函数。

功能	描述
PFRestart	如果路径在断电时被中断，那么则用PFRestart（断电重启）进行检查，并可能需要采取一些特定行动。该函数会在电流等级、基本等级或中断等级上检查涉事路径。

系统参数

断电功能中没有系统参数，不过不论您是否安装了任一选项，您都能使用参数*Store signal at power fail*。

有关更多信息，请参阅 技术参考手册 - 系统参数。

2.1.6.3 断电功能的示例

对被中断路径的测试

在断电后又恢复作业的情况下，如果路径上发生过断电，那么就会像本示例一样进行测试。

```
!Test if path was interrupted
IF PFRestart() = TRUE THEN
    SetDO do5,1;
ELSE
    SetDO do5,0;
ENDIF
```


2.1.7 工艺配套功能

2.1.7.1 概述

目的

可使用工艺配套功能提供的一些RAPID指令来创建工艺应用。其用例包括：

- 可将连续工艺应用中所用的模拟输出信号设置成与机器人工具中心接触点（TCP）的速度成正比。
- 被“程序停止”或“紧急停止”所停止的连续工艺应用可从其停止处继续运行。

其中包括

工艺配套功能包括：

- 数据类型restartdata。
- 用于设置输出信号的指令：TriggSpeed。
- 与重启有关的所用指令：TriggStopProc和StepBwdPath。

限制

当您拥有基本功能*Fixed Position Events*时，才能使用指令TriggSpeed。

2.1.7.2 RAPID组件

数据类型

此处简述了工艺配套功能所用的每种数据类型。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各种数据类型。

数据类型	描述
restartdata	restartdata可包含机器人移动的停止序列处的指定I / O信号（工艺信号）的前值和后值。 当自主开发的工艺指令发生程序停止或紧急停止后，restartdata与TriggStopProc会被一同用于保存重启所需的数据。

指令：

此处简述了工艺配套功能所用的每条指令。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各条指令。

指令	描述
TriggSpeed	TriggSpeed的作用是定义一个模拟输出信号（该信号与TCP速度成某个数值比）的设置状况。 TriggSpeed只能搭配选项Fixed Position Events使用。
TriggStopProc	TriggStopProc的作用是保存所有已用工艺信号的前值和后值。 当自主开发的工艺指令发生程序停止或紧急停止后，TriggStopProc与数据类型restartdata会被一同用于保存重启所需的数据。
StepBwdPath	StepBwdPath的作用是从一则“重启”事件例程开始沿机器人路径向后移动TCP。

函数

没有针对工艺配套功能的RAPID函数。

2.1.7.3 工艺配套功能的示例

与速度成正比的信号

本示例将控制胶量的模拟输出信号设置成与速度成正比。

为了对机器人的任何速度骤减作出时间上的补偿，相应的模拟输出信号glue_ao会在TCP速度骤减前受到0.04秒的影响。如果glue_ao中计算出的逻辑模拟输出值溢出，那么便设置数字输出信号glue_err。

```
VAR triggdata glueflow;

!The glue flow is set to scale value 0.8 0.05 s before point p1
TriggSpeed glueflow, 0, 0.05, glue_ao, 0.8 \DipLag:=0.04,
  \ErrDO:=glue_err;
TriggL p1, v500, glueflow, z50, gun1;

!The glue flow is set to scale value 1 10 mm plus 0.05 s before
  point p2
TriggSpeed glueflow, 10, 0.05, glue_ao, 1;
TriggL p2, v500, glueflow, z10, gun1;

!The glue flow ends (scale value 0) 0.05 s before point p3
TriggSpeed glueflow, 0, 0.05, glue_ao, 0;
TriggL p3, v500, glueflow, z50, gun1;
```



提示

注意也可用NOSTEPIN例程概念来创建关于TriggSpeed的自主开发工艺指令。

在停止后恢复信号

在本例中，一个输出信号在程序停止或紧急停止后恢复了其数值。

无返回值程序supervise被定义为一则“通电”事件例程，而resume_signals则被定义为一则“重启”事件例程。

```
PERS restartdata myproc_data :=
  [FALSE,FALSE,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
...
PROC myproc()
  MoveJ p1, vmax, fine, my_gun;
  SetDO do_close_gun, 1;
  MoveL p2,v1000,z50,my_gun;
  MoveL p3,v1000,fine,my_gun;
  SetDO do_close_gun, 0;
ENDPROC
...
PROC supervise()
  TriggStopProc myproc_data \DO1:=do_close_gun, do_close_gun;
ENDPROC
```

下一页继续

2 RobotWare-OS

2.1.7.3 工艺配套功能的示例

续前页

```
PROC resume_signals()  
  IF myproc_data.preshadowval = 1 THEN  
    SetDO do_close_gun,1;  
  ELSE  
    SetDO do_close_gun,0;  
  ENDIF  
ENDPROC
```

向后移动TCP

在本例中，TCP在1秒内沿重启前的同一路径向后移动了30毫米。

无返回值程序move_backward被定义为一则“重启”事件例程。

```
PROC move_backward()  
  StepBwdPath 30, 1;  
ENDPROC
```

2.1.8 中断功能

2.1.8.1 概述

目的

除RAPID中始终包括的中断工件外，Advanced RAPID的中断功能还有一些额外的工件。基本中断功能方面的更多信息请参见技术参考手册 - *RAPID*语言概览。

中断应用（Advanced RAPID）中的一些示例可在以下方面提供帮助：

- 在某永久变量改变数值时生成一次中断。
- 在发生一次错误时生成一次中断，然后查找该错误方面的更多信息。

其中包括

Advanced RAPID的中断功能包括：

- 错误中断的数据类型：trapdata、errdomain、errtype。
- 用于生成中断的指令：IPers和IError。
- 用于在错误中断方面查找更多信息的指令：GetTrapData和ReadErrData。

2.1.8.2 RAPID组件

数据类型

此处简述了中断功能所用的每种数据类型。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各种数据类型。

数据类型	描述
trapdata	trapdata代表了与“导致了当前待执行软中断例程的那次中断”有关的内部信息。
errdomain	errdomain的作用是指定一个错误域。系统会根据错误性质的差异而记录在不同的域内。
errtype	errtype的作用是指定一种错误类型（错误、警告、状态变化）。

指令：

此处简述了中断功能所用的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
IPers	IPers（永久中断）的作用是下令每当某永久变量发生改变时就生成一次中断。
IError	IError（中断错误）的作用是下令每当发生一次错误时就生成一次中断。
GetTrapData	GetTrapData的作用是用指令IError生成一则软中断例程。GetTrapData会获得“导致了待执行软中断例程的那次中断”方面的所有信息。
ReadErrData	ReadErrData的作用是用指令IError生成软中断例程。ReadErrData会读取GetTrapData获得的相关信息。
ErrRaise	ErrRaise的作用是在相关程序中创建一个错误，然后为相关例程的错误处理器创建一次相应的调用。也可在错误处理器中用ErrRaise来把当前错误传递给调用例程的错误处理器。

函数

没有针对中断功能的RAPID函数。

2.1.8.3 中断功能的示例

当永久变量发生变化时中断

在本例中，系统在永久变量counter的数值发生变化时调用了一则软中断例程。

```
VAR intnum int1;
PERS num counter := 0;

PROC main()
  CONNECT int1 WITH iroutinel;
  IPers counter, int1;
  ...
  counter := counter + 1;
  ...
  Idelete int1;
ENDPROC

TRAP iroutinel
  TPWrite "Current value of counter = " \Num:=counter;
ENDTRAP
```

错误中断

在本例中，系统在发生一次错误时调用了一则软中断例程。该软中断例程决定了相应的错误域和错误编号，并通过输出信号向外传输了这些内容。

```
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;
  ...
  a:=3;
  b:=0;
  c:=a/b;
  ...
  IDelete err_interrupt;
ENDPROC

TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
  SetGO go_err1, err_domain;
  SetGO go_err2, err_number;
ENDTRAP
```

2.1.9 用户消息功能

2.1.9.1 概述

目的

用户消息功能的作用一是设置事件编号，二是加快处理事件消息以及相关用户界面呈现的其它文本。

此处是一些应用示例：

- 从简化了更新和转换内容的文本表格文件中获取用户消息。
- 在“提升”指令中添加任何将被作为错误恢复常量的系统错误编号以及“错误”处理器测试所需的系统错误编号。

其中包括

用户消息功能包括：

- 文本表格操作指令TextTabInstall。
- 文本表格操作指令：TextTabFreeToUse、TextTabGet和TextGet。
- 处理错误编号的指令：BookErrNo。

2.1.9.2 RAPID组件

数据类型

没有针对用户消息功能的RAPID数据类型。

指令：

此处简述了用户消息功能所用的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
BookErrNo	BookErrNo的作用是定一个新的RAPID系统错误编号。
TextTabInstall	TextTabInstall的作用是在系统中安装一份文本表格。

函数

此处简述了用户消息功能所用的每则函数。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各则函数。

功能	描述
TextTabFreeToUse	TextTabFreeToUse的是测试能否自由使用相关的文本表格名称（尚未安装在系统中）。
TextTabGet	TextTabGet的作用是从一份由用户定义的文件表格的文件表格编号。
TextGet	TextGet的作用是从系统文件表格中获取一段文本字符串。

2.1.9.3 用户消息功能的示例

卷册错误编号

该示例展示了如何添加一个新的错误编号。

```
VAR intnum siglint;

!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;

PROC main()
  !Book the new RAPID system error number
  BookErrNo ERR_GLUEFLOW;

  !Raise glue flow error if dil=1
  IF dil=1 THEN
    RAISE ERR_GLUEFLOW;
  ENDIF
ENDPROC

!Error handling
ERROR
IF ERRNO = ERR_GLUEFLOW THEN
  ErrWrite "Glue error", "There is a problem with the glue flow";
ENDIF
```

文本表格文件的错误消息

该示例展示了如何从一份文本表格文件中获取用户消息。

名为HOME:/language/en/text_file.xml的文件中有一份名为text_table_name的文本表格。该表格包含了用英语表述的错误消息。

在“通电”事件时执行无返回值程序install_text。第一次执行时会安装文本表格文件text_file.xml，而再次执行时函数TextTabFreeToUse则会返回FALSE，且不会重复安装。

之后会用该表格来获取用户界面消息。

```
VAR num text_res_no;

PROC install_text()
  !Test if text_table_name is already installed
  IF TextTabFreeToUse("text_table_name") THEN
    !Install the table from the file HOME:/language/en/text_file.xml
    TextTabInstall "HOME:/language/en/text_file.xml";
  ENDIF
  !Assign the text table number for text_table_name to text_res_no
  text_res_no := TextTabGet("text_table_name");
ENDPROC

...
!Write error message with two strings from the table text_res_no
ErrWrite TextGet(text_res_no, 1), TextGet(text_res_no, 2);
```

2.1.9.4 文本表格文件

概述

一份XML文件保存有一份文本表格（每份文件能包含采用一种语言的一份表格），而该表格则能包含任意数目的文本字符串。

对文本表格的说明

此处描述了相关文本表格文件中的XML标签和自变数。

标签	变元	描述
Resource		代表一份文本表格。一份文件只能包含一个Resource实例。
	Name	相关文本表格的名称。通过RAPID指令TextTabGet使用。
	Language	针对文本字符串所用语言的语言代码。 当前未使用该自变数。RAPID指令TextTabInstall只能处理英语文本。
Text		代表了一段文本字符串。
	Name	表格中的文本字符串编号。
Value		待使用的文本字符串。
Comment		对相关文本字符串及其用法的意见。

文本表格文件的示例

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Resource Name="text_table_name" Language="en">
  <Text Name="1">
    <Value>This is a text that is </Value>
    <Comment>The first part of my text</Comment>
  </Text>
  <Text Name="2">
    <Value>displayed in the user interface.</Value>
    <Comment>The second part of my text</Comment>
  </Text>
</Resource>
```

2.1.10 RAPID配套功能

2.1.10.1 概述

目的

RAPID配套功能由各式各样的例程组成，这些例程或许能为高级机器人程序员提供帮助。

此处是一些应用示例：

- 激活一件新的工具、工作对象或净负荷。
- 查找在当前例程外调用了哪个自变数。
- 测试上一次程序停止期间是否移动过程序指针。

其中包括

RAPID配套功能包括：

- 用于激活指定系统数据的指令：`SetSysData`。
- 获取原始数据对象名称的函数：`ArgName`。
- 针对程序指针移动相关信息的函数：`IsStopStateEvent`。

2.1.10.2 RAPID组件

数据类型

没有针对RAPID配套功能的数据类型。

指令

此处简述了RAPID配套功能所用的每条指令。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各条指令。

指令	描述
SetSysData	SetSysData激活（或更改当前激活的）机器人工具、机器人工作对象或机器人净负荷。

函数

此处简述了RAPID配套功能所用的每则函数。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各则函数。

功能	描述
ArgName	ArgName的作用是获取当前自变数或当前数据所需的原始数据对象的名称。
IsStopStateEvent	IsStopStateEvent会返回程序指针的移动信息。

2.1.10.3 RAPID配套功能的示例

激活工具

此例展示了如何激活一件已知工具：

```
!Activate tool1  
SetSysData tool1;
```

此例展示了当该工具的名称仅出现于字符串中时，该如何激活一件工具。

```
VAR string tool_string := "tool2";  
!Activate the tool specified in tool_string  
SetSysData tool0 \ObjectName := tool_string;
```

获取自变数的名称

本例中取用了par1的原始名称。输出内容将为“Argument name my_nbr with value 5”。

```
VAR num my_nbr :=5;  
procl my_nbr;  
  
PROC procl (num par1)  
  VAR string name;  
  name:=ArgName(par1);  
  TPWrite "Argument name "+name+" with value " \Num:=par1;  
ENDPROC
```

测试是否移动过程序指针

此例测试了上一次程序停止期间是否移动过程序指针。

```
IF IsStopStateEvent (\PPMoved) = TRUE THEN  
  TPWrite "The program pointer has been moved.";  
ENDIF
```

2.2 Analog Signal Interrupt

2.2.1 Analog Signal Interrupt介绍

目的

Analog Signal Interrupt的用途一是监管一个模拟信号，二是在达到某指定值时生成一次中断。

与各种轮询法相比，Analog Signal Interrupt更快捷，更易于执行，对计算机能力的需求也更低。

此处是一些应用示例：

- 用更好的定时来节省周期时间（在某信号达到指定值时——而非等候轮询——就准时开始移动机器人）。
- 如果某信号值超出了其允许范围，则会显示警报或错误消息。
- 如果某信号值达到了某种危险水平，则会停止相应的机器人。

其中包括

您可通过RobotWare基本功能Analog Signal Interrupt来访问以下指令：

- ISignalAI
- ISignalAO

基本方法

这是Analog Signal Interrupt的一般用法。[第49页的代码示例](#)用一个更详细的示例展示了其具体用法。

- 1 创建一则软中断例程
- 2 用指令CONNECT来连接相关的软中断例程。
- 3 用指令ISignalAI或ISignalAO来定义相应的中断条件。

限制

如果您有一个工业网络选项（比如DeviceNet或PROFIBUS），那么就只能使用模拟信号。

2.2.2 RAPID组件

数据类型

Analog Signal Interrupt中不包括任何数据类型。

指令：

此处简述了Analog Signal Interrupt中的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
ISignalAI	为应调用的一则中断例程定义一个模拟输入信号的相关数值。 可设置成当相关信号值高于或低于某指定值时就出现一次中断，或设置成当处于或超出某指定范围时就出现一次中断。此外还可指定该中断是仅出现一次还是反复出现。
ISignalAO	为应调用的一则中断例程定义一个模拟输出信号的相关数值。 可设置成当相关信号值高于或低于某指定值时就出现一次中断，或设置成当处于或超出某指定范围时就出现一次中断。此外还可指定该中断是仅出现一次还是反复出现。

函数

Analog Signal Interrupt中不包括RAPID函数。

2.2.3 代码示例

温度监视

本例中的一部温度传感器与信号ail相连。

含一条警告的一则中断例程被设置成“在120度到130度内，温度每上升0.5度就执行一次”。另一则停止相关机器人的软中断例程则被设置成“在温度升至130度以上后就尽快执行”。

```
VAR intnum ail_warning;
VAR intnum ail_exceeded;

PROC main()
  CONNECT ail_warning WITH temp_warning;
  CONNECT ail_exceeded WITH temp_exceeded;
  ISignalAI ail, AIO_BETWEEN, 130, 120, 0.5, \DPos, ail_warning;
  ISignalAI \Single, ail, AIO_ABOVE_HIGH, 130, 120, 0, ail_exceeded;
  ...
  IDelete ail_warning;
  IDelete ail_exceeded;
ENDPROC

TRAP temp_warning
  TPWrite "Warning: Temperature is "\Num:=ail;
ENDTRAP

TRAP temp_exceeded
  TPWrite "Temperature is too high";
  Stop;
ENDTRAP
```

2.3 Auto Acknowledge Input

描述

*Auto Acknowledge Input*是一个系统输入项，当用户用机器人控制器上的钥匙开关将操作员手动模式切换为自动模式后，该输入项便会接受FlexPendant上展示的对话。



警告

请注意，使用此类输入项将违反相关安全标准ISO 10218-1第5.3.5章“单控制点”（Single point of control）中的规定。其规定内容如下：

“所设计和构建的机器人控制系统应满足以下要求：当机器人处于本地悬吊控制器或其它教学装置的控制之下时，没有任何其它控制源能让该机器人开始运动或更改本地控制选择。”

由此可见，用户绝对有必要采用其它安全手段来继续满足相关标准与机械指令的各项要求，同时也绝对有必要对整个围笼进行风险评估。系统集成人员将负责此类额外安排和风险评估，另外除非已完成了这些行动，否则不得启用相关系统。

限制

不能用FlexPendant或RobotStudio来定义该系统参数（只能用I / O配置文件中的一段文本字符串）。

激活Auto Acknowledge Input

用以下无返回值程序来激活*Auto Acknowledge Input*所需的系统输入项。

	操作
1	用FlexPendant或RobotStudio保存I / O配置文件 <i>eio.cfg</i> 的一份副本。
2	用一个文本编辑器编辑I / O配置文件 <i>eio.cfg</i> 在组SYSSIG_IN中添加以下行： -Signal "my_signal_name" -Action "AckAutoMode" 宜作为系统输入项的已配置数字输入信号被命名为my_signal_name。
3	保存该文件，然后在控制器上重新加载该文件。
4	重启系统，以此激活该信号。

2.4 Cyclic bool

2.4.1 循环评估逻辑条件

目的

循环评估逻辑条件*Cyclic bool*的目的是允许RAPID程序员能将逻辑条件与永久布尔变量关联起来。逻辑条件每12 ms评估一次，结果将写入关联的变量。

其中包括

RobotWare基础功能*Cyclic bool*包括：

- 设置*Cyclic bool*的指令：SetupCyclicBool、RemoveCyclicBool、RemoveAllCyclicBool
- 获取*Cyclic bool*状态的函数：GetMaxNumberOfCyclicBool、GetNextCyclicBool、GetNumberOfCyclicBool。

基本方法

这是使用*Cyclic bool*的常规方法。有关如何执行的详细示例，请参阅第54页的*Cyclic bool*示例。

- 1 声明一个永久布尔变量，例如：

```
PERS bool cyclicbool1;
```

- 2 将一个逻辑条件关联到变量，例如：

```
SetupCyclicBool cyclicbool1, doSafetyIsOk = 1;
```

- 3 编程时使用此变量，例如：

```
WHILE cyclicbool1 = 1 DO
    ! Do what's only allowed when all safety is ok
    ...
ENDWHILE
```

- 4 当不再使用时取消关联，例如：

```
RemoveCyclicBool cyclicbool1;
```

语法

SetupCyclicBool Flag Cond

Flag应为：

- 数据类型：bool
 - 对象类型：PERS或TASK PERS

Cond应为一个bool表达式，其中可以包含：

- 数据类型：num、dnum和bool
 - 对象类型：PERS、TASK PERS或CONST
- 数据类型：signal_{di}、signal_{do}或物理数字输入(DI)和数字输出(DO)
 - 对象类型：VAR
- 运算符：‘NOT’、‘AND’、‘OR’、‘XOR’、‘=’、‘(’、‘)’

下一页继续

2 RobotWare-OS

2.4.1 循环评估逻辑条件

续前页

RemoveCyclicBool Flag

Flag应为：

- 数据类型：bool
 - 对象类型：PERS或TASK PERS

限制

- 逻辑条件中不允许使用记录和数组。
- 同时可以关联最多60个条件。

2.4.2 RAPID组件

关于RAPID组件

这是在*Cyclic bool*中所有RAPID指令、函数与数据类型的概述。

有关更多信息，请参阅 技术参考手册 - *RAPID*指令、函数和数据类型。

指令：

指令	描述
SetupCyclicBool	SetupCyclicBool关联一个逻辑条件到一个布尔变量。
RemoveCyclicBool	RemoveCyclicBool去除一个关联的逻辑条件。
RemoveAllCyclicBool	RemoveAllCyclicBool去除全部关联的逻辑条件。

函数

功能	描述
GetMaxNumberOfCyclicBool	GetMaxNumberOfCyclicBool获取可以同时关联的循环评估逻辑条件的最大数量。
GetNextCyclicBool	GetNextCyclicBool获取一个关联的循环评估逻辑条件的名称。
GetNumberOfCyclicBool	GetNumberOfCyclicBool获取一个关联的循环评估逻辑条件的编号。

数据类型

*Cyclic bool*不包含数据类型。

2.4.3 Cyclic bool示例

使用数字输入和输出信号

```
! Wait until all signals are set
PERS bool cyclicbool1 := FALSE;

PROC main()
  SetupCyclicBool cyclicbool1, di1=1 AND do2=1;
  WaitUntil cyclicbool1=TRUE;
  ! All is ok
  ...
  ! Remove connection when no longer in use
  RemoveCyclicBool cyclicbool1;
ENDPROC
```

使用bool变量

```
! Wait until all flags are TRUE
PERS bool cyclicbool1 := FALSE;
TASK PERS bool flag1 := FALSE;
PERS bool flag2 := FALSE;

PROC main()
  SetupCyclicBool cyclicbool1, flag1=TRUE AND flag2=TRUE;
  WaitUntil cyclicbool1=TRUE;
  ! All is ok
  ...
  ! Remove connection when no longer in use
  RemoveCyclicBool cyclicbool1;
ENDPROC
```

使用num和dnum变量

```
! Wait until all conditions are met
PERS bool cyclicbool1 := FALSE;
PERS bool cyclicbool2 := FALSE;
PERS num num1 := 0;
PERS dnum1 := 0;

PROC main()
  SetupCyclicBool cyclicbool1, num1=7 OR dnum1=10000000;
  SetupCyclicBool cyclicbool2, num1=8 OR dnum1=11000000;
  WaitUntil cyclicbool1=TRUE;
  ...
  WaitUntil cyclicbool2=TRUE;
  ...
  ! Remove all connections when no longer in use
  RemoveAllCyclicBool;
ENDPROC
```

使用alias变量

```
! Wait until all conditions are met
ALIAS bool aliasBool;
ALIAS num aliasNum;
ALIAS dnum aliasDnum;

PERS bool cyclicbool1 := FALSE;
PERS aliasBool flag1 := FALSE;
PERS aliasNum num1 := 0;
PERS aliasDnum dnum1 := 0;

PROC main()
  SetupCyclicBool cyclicbool1, flag1=TRUE AND (num1=7 OR
    dnum1=10000000);
  WaitUntil cyclicbool1=TRUE;
  ! All is ok
  ...
  ! Remove connection when no longer in use
  RemoveCyclicBool cyclicbool1;
ENDPROC
```

使用用户定义的常量进行比较

```
! Wait until all conditions are met
PERS bool cyclicbool1;
PERS bool flag1 := FALSE;
PERS num num1 := 0;
PERS dnum dnum1 := 0;
CONST bool MYTRUE := TRUE;
CONST num NUMLIMIT := 10;
CONST dnum DNUMLIMIT := 10000000;

PROC main()
  SetupCyclicBool cyclicbool1, flag1=MYTRUE AND num1=NUMLIMIT AND
    dnum1=DNUMLIMIT;
  WaitUntil cyclicbool1=TRUE;
  ! All is ok
  ...
  ! Remove connection when no longer in use
  RemoveCyclicBool cyclicbool1;
ENDPROC
```

通过引用传递参数

如果在一个调用的无返回值程序中使用了指令SetupCyclicBool，则有可能将条件作为该无返回值程序的参数传递。

使用通过引用传递的条件仅对SetupCyclicBool有效。通过引用传递的条件与SetupCyclicBool的条件有相同限制。.

无论模块是Nostepin或有任何其他模块属性，此功能均有效。

```
MODULE MainModule
CONST robtarget p10 := [[600,500,225.3], [1,0,0,0], [1,1,0,0],
                        [11,12.3,9E9,9E9,9E9,9E9]];
PERS bool m1;
PERS bool Flag2 := FALSE;

PROC main()
  ! The Expression (di_1 = 1) OR Flag2 = TRUE shall be used by
    SetupCyclicBool
  my_routine (di_1 = 1) OR Flag2 = TRUE;
ENDPROC

PROC my_routine(bool X)
  ! It is possible to pass arguments between several procedures
  MySetCyclicBool X;
ENDPROC

PROC MySetCyclicBool (bool Y)
  RemoveCyclicBool m1;
  ! Only SetupCyclicBool can pass arguments
  SetupCyclicBool m1, Y;
  ! If conditions passed by reference shall be used by any other
    instruction, the condition must be setup with
    SetupCyclicBool before it can be used.
  WaitUntil m1;
  MoveL p10, v1000, z30, tool2;
ENDPROC
ENDMODULE
```


2.5 Electronically Linked Motors

2.5.1 概述

描述

Electronically Linked Motors用于实现各电机的主动 / 从动配置（比如两根附加轴）。从动轴的位置、速度和加速度会始终遵从主动轴的位置、速度和加速度。如果主动件与从动件之间采用了刚性机械连接，便可使用扭矩从动函数。该功能并非把主动件和从动件的位置调整得如出一辙，二是在各轴之间分配扭矩。主动件和从动件之间会出现少许位置误差，这具体取决于反冲和机械错配的情况。

目的

Electronically Linked Motors的主要用途是更换龙门式机器的驱动轴，但也可用基本功能来控制任何其它电机组。

其中包括

您可通过RobotWare基本功能Electronically Linked Motors来访问：

- 用于定义联动电机组和微调轴位置的一则服务程序
- 用于配置一根从动轴的系统参数

基本方法

这是设置Electronically Linked Motors的一般方式。更详细的具体做法请参见相应章节。

- 1 配置您想使用的附加轴。请参见应用手册 - *Additional axes and stand alone controller*。
- 2 配置类型`Linked M Process`、`Process`和`Joint`下的系统参数容限。
- 3 重启控制器，从而使所做改动生效。
- 4 将数值设置成数据变量，从而定义联动电机组和连接从动轴与主动轴。
- 5 用该服务程序来微调位置，或在出现位置误差后用其重启从动件。

限制

最多能有5根从动轴。用户既可配置成每根从动轴各跟随一根主动轴，也可配置成若干从动轴跟随一根主动轴，但从动轴的总数不能超过5根。

ABB机器人（IRB机器人）不能作为从动轴。主动轴既可以是一根附加轴，也可以是一根机器人轴。

只有当从动轴与主动轴连接在同一个驱动模块上时，才能使用扭矩从动函数。

使用扭矩从动功能可能会减少从动轴的数目，这具体取决于配置了主动轴的驱动模块有多少根可以使用的轴。

RAPID指令`IndReset` (*Independent Reset*) 不能与Electronically Linked Motors一同使用。

2 RobotWare-OS

2.5.2.1 系统参数

2.5.2 配置

2.5.2.1 系统参数

关于系统参数

此处简述了Electronically Linked Motors所用的每个参数。更多信息请参见技术参考手册 - 系统参数中的各个参数。

Joint

这些参数属于主题*Motion*和类型*Joint*之列。

参数	描述
Follower to Joint	指定该轴应跟随哪根主动轴。请引用类型 <i>Joint</i> 中的参数 <i>Name</i> 。机器人轴是指下划线部分和轴编号之后的rob1（比如rob1_6）。
Use Process	所调用工艺的Id名称。请引用类型 <i>Process</i> 中的参数 <i>Name</i> 。
Lock Joint in Ipol	一个旗标，该旗标锁定了相关轴，使得系统无法在路径插补时使用该轴。如果该轴正以电子化方式链接到另一根轴上，那么该参数必须设置成TRUE。

Process

这些参数属于主题*Motion*和类型*Process*之列。

参数	描述
Name	该工艺的Id名称。
Use Linked Motor Process	电子链接电机工艺的Id名称。请引用类型 <i>Linked M Process</i> 中的参数 <i>Name</i> 。

Linked M Process

这些参数属于主题*Motion*和类型*Linked M Process*之列。

参数	描述
Name	所链接电机工艺的Id名称。
Offset Adjust Delay Time	从动件开始跟随主动件之前的控制延时。 该参数的作用是给主动件一些时间，以让其在从动件开始跟随前达到稳定。
Max Follower Offset	主动件与从动件的间距（以弧度或米为单位）的最大允许差值。 如果超出了 <i>Max Follower Offset</i> ，则会启动紧急停止。
Max Offset Speed	主动件与从动件间的相对速度（以弧度 / 秒或米 / 秒为单位）的最大允许差值。 如果超出了 <i>Max Offset Speed</i> ，则会启动紧急停止。
Offset Speed Ratio	定义 <i>Max Offset Speed</i> 中有多大部分能用来补偿位置误差。
Ramp Time	加速至 <i>Max Offset Speed</i> 所需的时间。 位置调节所需的比例常数在 <i>Ramp Time</i> 期间从零增大至其最终值（ <i>Master Follower kp</i> ）。
Master Follower kp	位置调节所需的比例常数。确定位置误差的补偿速度有多快。
Torque follower	如果最好让主动件和从动件共享扭矩，而非调节确切位置，那么就最好设置成True。 只有当从动轴与主动轴连接在同一个驱动模块上时，才能使用该参数。

下一页继续

参数	描述
Torque distribution	最好将该（总扭矩的）比率用在从动件上（比如0.3意味着30%在从动件上，70%在主动件上）。如果驱动器与电机彼此相当，那么就通常设置成0.5。
Follower axis pos. acc. reduction	设置该数值的作用是降低从动件位置环路的准确性。如果刚性机械连接中存在较大的位置错配，导致机械结构在各电机之间形成了较大的扭矩，那么就会需要这种功能。 <ul style="list-style-type: none">• 0：未启用“降低准确度”功能• 10-30典型值

2 RobotWare-OS

2.5.2.2 配置示例

2.5.2.2 配置示例

关于此例

此例展示了如何将附加轴M8DM1配置成轴M7DM1和M9DM1的从动件，以及又如何将后两轴配置成机器人轴6的从动件。

Joint

Name	Follower to Joint	Use Process	Lock Joint in Ipol
M7DM1			
M8DM1	M7DM1	ELM_1	True
M9DM1	rob1_6	ELM_2	True

Process

Name	Use Linked Motor Process
ELM_1	Linked_m_1
ELM_2	Linked_m_2

Linked M Process

Name	Offset Adjust Delay Time	Max Follower Offset	Max Offset Speed	Offset Speed Ratio	Ramp Time	Master Follower kp
Linked_m_1	0.2	0.05	0.05	0.33	1	0.05
Linked_m_2	0.1	0.1	0.1	0.4	1.5	0.08

2.5.3 管理一根从动轴

2.5.3.1 使用服务程序

关于服务程序

该服务程序用于以下方面：

- 校准从动轴
- 在出现位置误差后重置从动轴
- 微调一根扭矩从动轴，具体请参见[第65页的微调一个扭矩从动件](#)。

数据变量

在启动时，服务例程将从系统参数中读取数值，然后为该例程使用的一组数据变量设置相应数值。如果出现问题，则仅需手动设置这些变量即可。具体请参见[第68页的数据设置](#)。

启动服务程序



注意

必须将控制器置于手动或自动模式后才能运行该服务程序。

步骤	操作
1	在程序视图中轻击调试，然后选择调用例程...
2	选择Linked_m，然后轻击转至。
3	按下“运行（RUN）”按钮来启动该服务程序。 该服务程序会显示在屏幕上。
4	轻击菜单1。 系统中设置的从动轴会显示在任务栏上。
5	轻击想运用该服务程序的从动轴。 此时会显示该服务程序的主菜单。

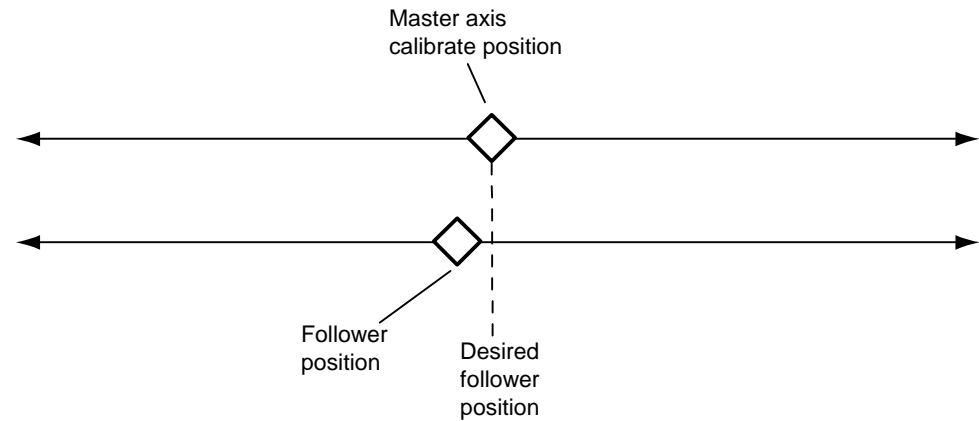
菜单按钮

Button	描述
自动	会自动把从动轴移到主动轴所对应的位置上，具体请参见 第64页的自动重置从动件 。
停止 (STOP)	停止移动从动轴。可在点动或自动时使用，同时必须立即停止相关移动。
点动	从动轴的手动逐步移动，具体请参见 第62页的点动从动轴 。 如果从动轴正与主动轴保持同步，那么当您轻击自动或推出该服务程序时，从动轴便会恢复其位置。
UNSYNC	用于暂停从动轴与主动轴之间的同步，具体请参见 第62页的解除同步 。
帮助	显示该服务程序的使用帮助信息。按钮下一则用于显示下一则帮助主题。

2.5.3.2 校准从动轴位置

概述

您必须先定义主动件和从动件的校准位置，然后从动轴才能跟随主动轴。



en0400000963

按以下无返回值程序来进行此次校准：

- 1 将主动轴点动至其校准位置。
- 2 解除从动轴与主动轴之间的同步。具体请参见[第62页的解除同步](#)。
- 3 将从动件点动至所需位置。具体请参见[第62页的点动从动轴](#)。
- 4 精细校准从动轴。具体请参见[第62页的精细校准](#)。

解除同步

步骤	操作
1	在该服务程序的主菜单中轻击解除同步。
2	通过轻击是来确认您想解除各轴的同步。
3	当有信息文本要求您重启控制器时，便重启控制器。 重启后从动轴便不再与主动轴同步。

点动从动轴

步骤	操作
1	在该服务程序的主菜单中轻击点动。
2	选择您点动时的从动轴移动速度。
3	选择您点动时的从动轴每移动一步的步长。
4	轻击正或负，具体取决于您想往哪个方向移动从动轴。 点动从动轴，直至其准确抵达校准位置（主动轴校准位置所对应的位置）为止。

精细校准

步骤	操作
1	在ABB菜单上选择校准。
2	选择从动轴所属的机械单元。

下一页继续

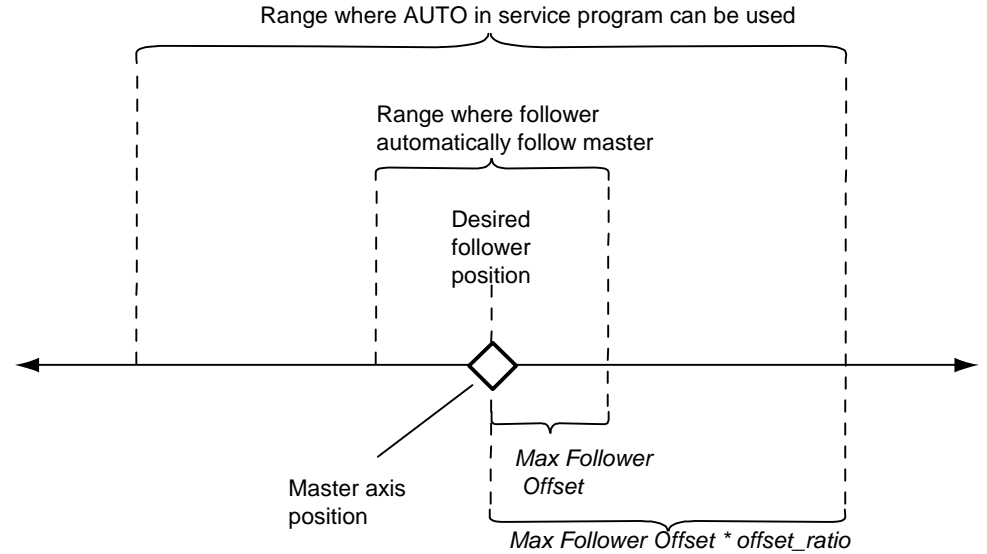
步骤	操作
3	轻击按钮校准参数。
4	轻击精细校准...
5	在出现的警告对话框中轻击是。
6	选择作为从动轴的轴，然后轻击校准。
7	在出现的警告对话框中轻击校准。 此时校准从动轴。从动轴一旦校准完毕，就会再次与主动轴同步。

2.5.3.3 重置从动轴

概述

如果从动件偏移量超过了其容限（用系统参数*Max follower offset*进行配置），那么就必须用该服务程序来把从动件移回容限之内。如果从动件位于“自动”范围内，那么该服务程序会自动完成这一操作。其它情况下就必须以手动方式点动从动轴。

系统参数*Max Follower Offset*与数据变量*offset_ratio*的乘积决定了可采用“自动”的范围。



en0400000962

自动重置从动件

步骤	操作
1	在该服务程序的主菜单中轻击自动。
2	选择从动轴向所需位置移动时的速度。

通过手动点动来重置从动件

步骤	操作
1	在该服务程序的主菜单中轻击点动。
2	选择您点动时的从动轴移动速度。
3	选择您点动时的从动轴每移动一步的步长。
4	轻击正或负，具体取决于您想往哪个方向移动从动轴。 点动从动轴，直至其进行 <i>Max Follower Offset</i> 的容限范围为止（或在足够接近时使用“自动”）。

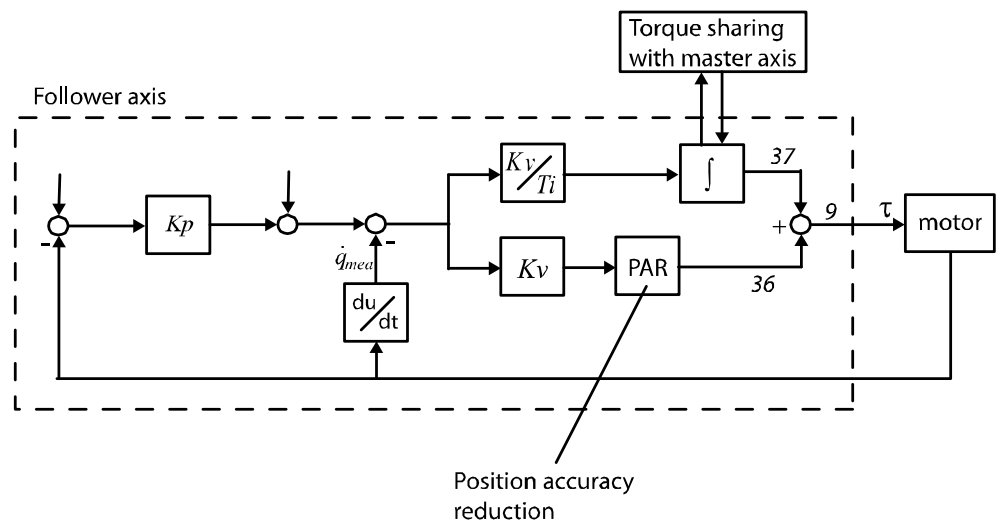
2.5.4 微调一个扭矩从动件

2.5.4.1 扭矩从动件的描述

关于扭矩从动件

可通过从动轴的设置来让主动件和从动件共享扭矩。只有当从动轴与主动轴连接在同一个驱动模块上时，才能进行这一操作。

以下是从动轴控制环路的简图。



en0900000679

扭矩分配

将在控制环路的主要部分上共享扭矩。若将扭矩分配设置成0.5，则主动件和从动件各占总扭矩的一半；若设置成0.3，则从动轴占总扭矩的30%，主动轴占总扭矩的70%。

降低定位准确度

如果相关机械结构既十分刚直，又存在机械错配或较大游隙，那么总扭矩的主要来源就是上述比例部分。如果因主动件与从动件之间扭矩差过大而产生问题，那么便可使用降低定位准确度函数（插图中的PAR）来降低从动轴抵达某一位置时的准确度，从而使从动件更像一个真实的扭矩从动件。

可用来检查相关行为的测试信号：

测试信号	测试信号编号
扭矩中的主要部分	37
扭矩中的比例部分	36
总扭矩参考值（也包括任何前馈扭矩）	9

2.5.4.2 使用服务程序

关于扭矩从动件的服务程序

在扭矩从动件服务程序中，这一部分的作用是为某些参数查找合适的值。一旦找到这些值，系统就会更新系统参数，并重新进行一次精细校准，然后便无需再对该扭矩从动件进行微调了。

打开微调扭矩从动件菜单

	操作	图示
1	启动服务程序（正如第61页的启动服务程序中的前几个步骤所述。	
2	轻击菜单2。	
3	轻击从动轴的名称来进行微调。	
4	按下文所述使用微调扭矩从动件菜单。	

微调扭矩分配

用该无返回值程序来更改主动轴与从动轴之间的扭矩分配。

	操作	图示
1	轻击扭矩分配。	
2	打一个数字（0到1之间）来作为该从动件在总扭矩中所占的份额。 举例来说，0.3将使总扭矩的30%位于该从动件上，70%位于主动件上。	
3	若要用此新值来更新相应的系统参数，则轻击保存为cfg。 如果不保存为cfg，则在机器人控制器重启前将一直使用该新值，但重启后便会丢失该值。	

微调位置准确度降幅

用该无返回值程序来设置扭矩从动轴的位置准确度降幅。

	操作	图示
1	轻击位置准确度降幅。	
2	输入一个数字来决定降低后的位置准确度。 0意味着不降低位置准确度。 若要降低主动件与从动件之间的扭矩张力，则通常把扭矩从动件设为10到30。	
3	若要用此新值来更新相应的系统参数，则轻击保存为cfg。 如果不保存为cfg，则在机器人控制器重启前将一直使用该新值，但重启后便会丢失该值。	

微调临时位置 Δ

用该无返回值程序来微调扭矩从动轴的位置 Δ ，然后用该 Δ 值来调节该从动轴的精细校准。

	操作	图示
1	轻击临时位置 Δ 。	
2	输入一个将添加到从动轴位置引用项中的数字（电机侧的度数）。	
3	测试哪个数值产生的扭矩张力最低，然后对主动轴做一次精细校准，从而用当前的位置 Δ 来更新相应的从动轴。	

2.5.5 数据设置

2.5.5.1 设置服务程序的数据

概述

当Electronically Linked Motors的服务例程刚刚启动时，系统会从所链接电机配置中读取一些数据变量，并交由服务程序使用。若未正确读取这些变量，则需在服务程序中进行编辑。

数据描述

数据变量	描述
l_f_axis_name	将显示在FlexPendant示教器上的从动轴名称。 有5个元素的字符串数组，每个元素对应一根从动轴。如果您仅链接了一台电机，则请只使用第一个元素。
l_f_mecunt_n	该从动轴的机械单元的名称。请引用类型 <i>Mechanical Unit</i> 中的系统参数 <i>Name</i> 。 有5个元素的字符串数组，每个元素对应一根从动轴。如果您仅链接了一台电机，则请只使用第一个元素。
l_f_axis_no	定义从动轴是该机械单元中的哪根轴（l_f_mecunt_n）。 有5个元素的数字数组，每个元素对应一根从动轴。如果您仅链接了一台电机，则请只使用第一个元素。
l_m_mecunt_n	该主动轴的机械单元的名称。请引用类型 <i>Mechanical Unit</i> 中的系统参数 <i>Name</i> 。 有5个元素的字符串数组，每个元素对应一根主动轴。如果您仅链接了一台电机，则请只使用第一个元素。
l_m_axis_no	定义主动轴是该机械单元中的哪根轴（l_m_mecunt_n）。 有5个元素的数字数组，每个元素对应一根主动轴。如果您仅链接了一台电机，则请只使用第一个元素。
offset_ratio	定义服务程序中“自动（AUTO）”函数重置从动轴时的范围。offset_ratio定义的该范围是从动件自动跟随主动件之范围（由参数 <i>Max Follow Offset</i> 定义）的数倍。 如果从动件的位置误差大于 <i>Max Follower Offset</i> * offset_ratio，那么就必需手动重置该从动件。更多信息请参见第64页的 重置从动轴 。
speed_ratio	定义从动轴在服务程序控制下的速度。指定该数值时不能超过允许的最大手动速度（即是说0.5相当于最大手动速度的一半）。 有20个元素的数字数组。元素1到5会把每根从动轴的速度定义为“极慢”，元素6到10会把速度定义为“慢”，元素11到15会把速度定义为“正常”，元素16到20会把速度定义为“快”。如果您仅链接了一台电机，则请只使用元素1、6、11和16。
displacement	定义在点动服务程序下的从动轴时，每轻击一下正或负所移动的从动轴距离。指定该数值时以度或米为单位（取决于从动轴是环形移动还是直线移动）。 有20个元素的数字数组。元素1到5会把每根从动轴的位移定义为“极短”，元素6到10会把位移定义为“短”，元素11到15会把位移定义为“正常”，元素16到20会把位移定义为“长”。如果您仅链接了一台电机，则请只使用元素1、6、11和16。

编辑数据变量

此处描述了如何设置FlexPendant示教器的数据变量值。

步骤	操作
1	在ABB菜单中选择程序数据。
2	选择字符串，然后轻击显示数据。
3	选择l_f_axis_name，然后轻击编辑数值。
4	轻击第一个元素。
5	轻击相应的行来进行编辑。。
6	输入您第一根从动轴的自定名称。
7	如果您有一根以上的从动轴，那么请在后续元素上重复第4到6步。
8	在l_f_mecunt_n和l_m_mecunt_n上重复第3到7步。
9	在程序数据菜单中选择数字，然后对l_f_axis_no、l_m_axis_no、offset_ratio、speed_ratio和位移重复步骤3到步骤7。

2.5.5.2 数据设置示例

关于此例

此例展示了如何设置两根从动轴的数据变量。第一根从动轴为M8C1B1，属于附加轴M7C1B1的从动件；第二根从动轴为M9C1B1，属于机器人轴6的从动件。

l_f_axis_name

演示轴	l_f_axis_name中的元素和数值
Follower 1	{1}: "follow_external"
Follower 2	{2}: "follow_axis6"
Follower 3	{3}: ""
Follower 4	{4}: ""
Follower 5	{5}: ""

l_f_mecunt_n

演示轴	l_f_mecunt_n中的元素和数值
Follower 1	{1}: "M8DM1"
Follower 2	{2}: "M9DM1"
Follower 3	{3}: ""
Follower 4	{4}: ""
Follower 5	{5}: ""

l_f_axis_no

演示轴	l_f_axis_no中的元素和数值
Follower 1	{1}: 1
Follower 2	{2}: 1
Follower 3	{3}: 0
Follower 4	{4}: 0
Follower 5	{5}: 0

l_m_mecunt_n

演示轴	l_m_mecunt_n中的元素和数值
Master 1	{1}: "M7DM1"
Master 2	{2}: "rob1"
Master 3	{3}: ""
Master 4	{4}: ""
Master 5	{5}: ""

下一页继续

l_m_axis_no

演示轴	l_m_axis_no中的元素和数值
Master 1	{1}: 1
Master 2	{2}: 6
Master 3	{3}: 0
Master 4	{4}: 0
Master 5	{5}: 0

offset_ratio

演示轴	offset_ratio中的元素和数值
Follower 1	{1}: 10
Follower 2	{2}: 15
Follower 3	{3}: 0
Follower 4	{4}: 0
Follower 5	{5}: 0

speed_ratio

演示轴	极慢	慢	正常	快
Follower 1	{1}: 0.01	{6}: 0.05	{11}: 0.2	{16}: 1
Follower 2	{2}: 0.01	{7}: 0.05	{12}: 0.2	{17}: 1
Follower 3	{3}: 0	{8}: 0	{13}: 0	{18}: 0
Follower 4	{4}: 0	{9}: 0	{14}: 0	{19}: 0
Follower 5	{5}: 0	{10}: 0	{15}: 0	{20}: 0

displacement

演示轴	极短	短	正常	长
Follower 1	{1}: 0.001	{6}: 0.005	{11}: 0.02	{16}: 0.1
Follower 2	{2}: 0.01	{7}: 0.1	{12}: 1	{17}: 10
Follower 3	{3}: 0	{8}: 0	{13}: 0	{18}: 0
Follower 4	{4}: 0	{9}: 0	{14}: 0	{19}: 0
Follower 5	{5}: 0	{10}: 0	{15}: 0	{20}: 0

2.6 Fixed Position Events

2.6.1 概述

目的

Fixed Position Events的作用是是确保在明确定义TCP位置的情况下执行一项程序例程。

如果用设置成`fine`的区域自变数来调用一条移动指令，那么一旦TCP抵达其目标点，就会执行下一项例程。如果用设置成距离（如`z20`）的区域自变数来调用一条移动指令，那么在TCP还未靠近目标点前就可能会执行下一项例程。出现这种差别的原因是执行RAPID指令和执行机器人移动之间存在延时。

若用设置成`fine`的区域来调用移动指令，则会减慢移动速度。而若Fixed Position Events，那么只要TCP位于TCP路径上的任一指定位置，便能在不减慢移动速度的情况下执行一则例程。

其中包括

您可通过RobotWare基本功能Fixed Position Events来访问：

- 用于定义一起位置事件的指令
- 移动机器人并同时执行位置事件的指令
- 在未首先定义位置事件前移动机器人并在通过目标点时调用某则无返回值程序的指令

基本方法

即可通过一则调用某则无返回值程序的简化指令来使用Fixed Position Events，也可按下列通用步骤来设置此类事件。至于更为详细的设置示例，则请参见[第75页的代码示例](#)。

- 1 声明该位置事件。
- 2 定义该位置事件：
 - 应发生之时（与目标点位置进行对比）
 - 应做之事
- 3 调用一条使用位置事件的移动指令。当TCP尽可能靠近所定义的目标点时，便会发生这一事件。

2.6.2 RAPID组件与系统参数

数据类型

此处简述了Fixed Position Events中的每种数据类型。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各种数据类型。

数据类型	描述
triggdata	triggdata的作用是保存一起位置事件的数据。 一起位置事件的具体形式既可以是设置一个输出信号，也可以是在机器人移动路径上的某特定位置处运行一则中断例程。 triggdata也包含何时应出现行动的信息，比如TCP何时处在已定义的距离（与目标点之间）上。 triggdata是一种非数值的数据类型。
triggios	triggios的作用是保存指令TriggLIos所用位置事件的数据。 triggios会用一个num值来设置一个输出信号的值。
triggiosdnum	triggiosdnum的作用是保存指令TriggLIos所用位置事件的数据。 triggiosdnum会用一个dnum值来设置一个输出信号的值。
triggstrgo	triggstrgo的作用是保存指令TriggLIos所用位置事件的数据。 triggstrgo会用一个stringdig值（含有一个数字的字符串）来设置一个输出信号的值。

指令：

此处简述了Fixed Position Events中的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
TriggIO	TriggIO定义了一个输出信号的设置以及设置该信号的时间。其定义被保存在类型triggdata的一个变量中。 TriggIO不但能把定义“在距离目标点特定距离（以毫米为单位）处或离开目标点特定时间后发出相关信号”，还能把信号设置在已定义的距离或时间点上（指与起始位置之间的距离或时间）。 若将距离设置成0（零），那么信号将被设置在TCP尽可能靠近目标点（拐角路径中间）之时。
TriggEquip	TriggEquip的作用类似于TriggIO，差别在于TriggEquip无法补偿外部设备的内部延时。 举例来说，涂胶枪的信号就必须设置一个极短的时间，且该时间必须在压出胶水并开始胶合之前。
TriggInt	TriggInt定义了何时运行一则中断例程。其定义被保存在类型triggdata的一个变量中。 TriggInt定义了应在与距离目标点（或起始位置）多远处（以毫米为单位）调用中断例程。若将爱那个距离设置成0（零），那么将在TCO尽可能靠近目标点（拐角路径中间）时发生中断。
TriggCheckIO	TriggCheckIO为一个输入或输出信号定义了一项测试以及测试时间。其定义被保存在类型triggdata的一个变量中。 TriggCheckIO定义了一项测试，以便将一个输入或输出信号与某一数值进行对比。调用了一则中断例程。作为一种选项，用户可以在发生中断时停止机器人的移动。 TriggCheckIO不但能把定义“在距离目标点特定距离（以毫米为单位）处或离开目标点特定时间后进行测试”，还能在已定义的距离或时间点上（指与起始位置之间的距离或时间）进行测试。 若将距离设置成0（零），那么将在TCP尽可能靠近目标点（拐角路径中间）时调用中断例程。

下一页继续

2 RobotWare-OS

2.6.2 RAPID组件与系统参数

续前页

指令	描述
TriggRampAO	TriggRampAO定义了一个模拟输出信号的增减率以及增减时间。其定义被保存在类型triggdata的一个变量中。 TriggRampIO定义了什么开始信号增减以及增减长度。
TriggL	TriggL是一条类似于MoveL的移动指令。除移动外，TriggL指令还能设置输出信号、运行中断例程以及在固定位置检查输入或输出信号。 TriggL最多会执行8起保存为triggdata的位置事件。必须在调用TriggL前就对此进行定义。
TriggC	TriggC是一条类似于MoveC的移动指令。除移动外，TriggC指令还能设置输出信号、运行中断例程以及在固定位置检查输入或输出信号。 TriggC最多会执行8起保存为triggdata的位置事件。必须在调用TriggC前就对此进行定义。
TriggJ	TriggJ是一条类似于MoveJ的移动指令。除移动外，TriggJ指令还能设置输出信号、运行中断例程以及在固定位置检查输入或输出信号。 TriggJ最多会执行8起保存为triggdata的位置事件。必须在调用TriggJ前就对此进行定义。
TriggLIOs	TriggLIOs是一条类似于MoveL的移动指令。除移动外，TriggLIOs指令还能在固定位置设置输出信号。 TriggLIOs形同TriggEquip与TriggL的组合，差别在于TriggLIOs最多能处理50起保存为数组（数据类型为triggios、triggiosdnum或triggstrgo）的位置事件。
MoveLSync	MoveLSync是一条直线移动指令，其作用是在拐角路径中间处调用一则无返回值程序。
MoveCSync	MoveCSync是一条环形移动指令，其作用是在拐角路径中间处调用一则无返回值程序。
MoveJSync	MoveJSync是一条联合移动指令，其作用是在拐角路径中间处调用一则无返回值程序。

函数

Fixed Position Events中不包括RAPID函数。

系统参数

此处简述了Fixed Position Events中的每个参数。更多信息请参见技术参考手册 - 系统参数中的各个参数。

参数	描述
Event Preset Time	TriggEquip利用了RAPID执行与机器人移动之间的延时（约70毫秒）。如果设备延时超过70毫秒，那么就配置Event preset time的方式来延长机器人移动的延时。 Event preset time属于主题Motion下的类型Motion System。

2.6.3 代码示例

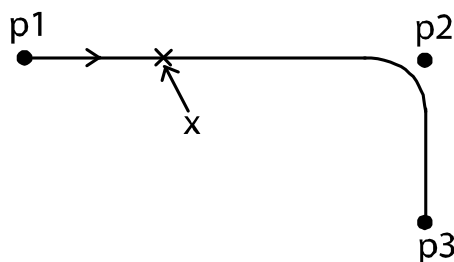
无Fixed Position Events的示例

在不使用Fixed Position Events的情况下，该代码可表达成：

```
MoveJ p1, vmax, fine, tool1;
MoveL p2, v1000, z20, tool1;
SetDO do1, 1;
MoveL p3, v1000, fine, tool1;
```

结果

该代码规定了TCP宜在设置do1前先抵达p2。由于机器人路径与执行指令之间存在延时，因此当TCP位于用X标记的位置时（参见插图），系统会设置do1。



xx0300000151

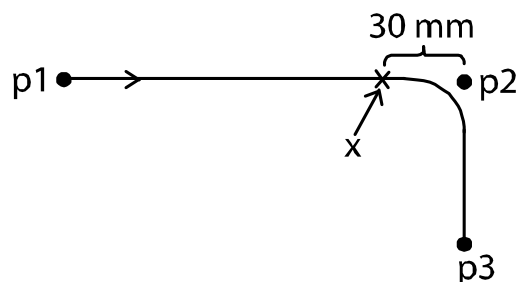
涉及TriggIO和TriggL指令的示例

可通过先定义位置事件、然后在系统执行该位置事件时移动机器人的方式来设置距离目标点30毫米处的输出信号。

```
VAR trigdata do_set;
!Define that do1 shall be set when 30 mm from target
TriggIO do_set, 30 \DOp:=do1, 1;
MoveJ p1, vmax, fine, tool1;
!Move to p2 and let system execute do_set
TriggL p2, v1000, do_set, z20, tool1;
MoveL p3, v1000, fine, tool1;
```

结果

将在TCP距离p230毫米处时设置信号do1。当TCP位于用X标记的位置时，系统会设置do1。



xx0300000158

下一页继续

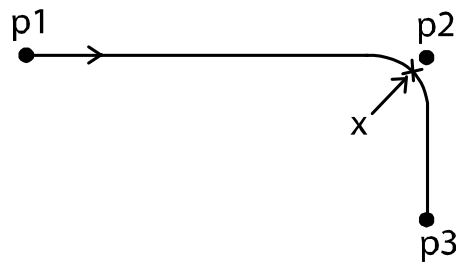
涉及MoveLSync指令的示例

可通过一次指令调用来实现“在机器人路径尽量靠近目标点时调用一则无返回值程序”。

```
MoveJ p1, vmax, fine, tool1;  
!Move to p2 while calling a procedure  
MoveLSync p2, v1000, z20, tool1, "proc1";  
MoveL p3, v1000, fine, tool1;
```

结果

当TCP位于用X标记的位置时（参见插图），系统便会调用该无返回值程序。



xx0300000165

2.7 File and Serial Channel Handling

2.7.1 File and Serial Channel Handling介绍

关于File and Serial Channel Handling

RobotWare基本功能File and Serial Channel Handling用RAPID代码为各种文件、现场总线和串行通道提供了机器人程序员控制能力。这能在以下等方面发挥作用：

- 用条码读取器读取。
- 将生产统计写入一份日志文件或写入一台打印机。
- 在机器人与一台个人电脑之间传输数据。

可将File and Serial Channel Handling中的这一功能划分为若干组：

功能组	描述
基于二进制和字符的通信	基本通信功能。采用基于二进制或字符的文件或串行通道的通信。
原始数据通信	包装在容器中的数据。尤其适用于现场总线通信。
文件与目录管理	浏览并编辑文件结构。

2.7.2 基于二进制和字符的通信

2.7.2.1 概述

目的

基于二进制和字符的通信的作用是：

- 将信息保存在远程内存或远程硬盘中
- 让机器人与其它装置进行通信

其中包括

为了处理基于二进制和字符的通信，您可通过RobotWare基本功能File and Serial Channel Handling来访问：

- 操纵一份文件或一条串行通道的指令
- 写入一份文件或一条串行通道的指令
- 读取一份文件或一条串行通道的指令
- 用于读取文件或串行通道的函数。

基本方法

这是基于二进制和字符的通信的一般用法。[第80页的代码示例](#)用一个更详细的示例展示了其具体用法。

- 1 打开一份文件或一条串行通道。
- 2 读取或写入相应的文件或串行通道。
- 3 关闭相应的文件或串行通道。

限制

不同的RAPID任务不能同时访问各文件、串行通道和现场总线。得通过基于二进制和字符的通信中的所有指令以及WriteRawBytes和ReadRawBytes来进行此类访问。举例来说，如果要在一项任务中执行一条ReadBin指令，那么就必须在另一项任务能执行WriteRawBytes前作好准备。

2.7.2.2 RAPID组件

数据类型

此处简述了基于二进制和字符的通信中所用的每种数据类型。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各种数据类型。

数据类型	描述
iodev	iodev包含了一份文件或一条串行通道的引用项。可通过指令Open将其与相关物理单元链接起来，然后用其进行读取和写入。

指令：

此处简述了基于二进制和字符的通信中使用的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
Open	Open的作用是打开一份文件或一条串行通道来进行读取或写入。
Close	Close的作用是关闭一份文件或一条串行通道。
Rewind	Rewind将该文件的位置设置在文件开头。
ClearIOBuff	ClearIOBuff的作用是清除一条串行通道的输入缓存。输入串行通道中的所有缓存字符都会被抛弃。
Write	Write的作用是写入一份基于字符的文件或一条串行通道。
WriteBin	WriteBin的作用是在一条二进制串行通道或一份文件中写入大量字节。
WriteStrBin	WriteStrBin的作用是在一条二进制串行通道或一份文件中写入一段字符串。
WriteAnyBin	WriteAnyBin的作用是在一条二进制串行通道或一份文件中写入任何类型。
ReadAnyBin	ReadAnyBin的作用是在一条二进制串行通道或一份文件中读取任何类型。

函数

此处简述了基于二进制和字符的通信中使用的每则函数。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各则函数。

功能	描述
ReadNum	ReadNum的作用是写入一份基于字符的文件或一条串行通道中读取一个数字。
ReadStr	ReadStr的作用是从一份基于字符的文件或一条串行通道中读取一段字符串。
ReadBin	ReadBin的作用是从一份文件或一条串行通道上读取一个字节（8位）。该函数可用于基于二进制和字符的文件或串行通道。
ReadStrBin	ReadStrBin的作用是从一条二进制串行通道或一份文件中读取一段字符串。

2.7.2.3 代码示例

用基于字符的文件进行通信

此例展示了如何读取和写入一份基于字符的文件。在FILE1.DOC中写入行“The number is :8”，然后读取FILE1.DOC的内容。先向FlexPendant示教器输出“The number is :8”，然后是“The number is 8”。

```
PROC write_to_file()
  VAR iodev file;
  VAR num number:= 8;
  Open "HOME:" \File:= "FILE1.DOC", file;
  Write file, "The number is :"\Num:=number;
  Close file;
ENDPROC

PROC read_from_file()
  VAR iodev file;
  VAR num number;
  VAR string text;

  Open "HOME:" \File:= "FILE1.DOC", file \Read;
  TPWrite ReadStr(file);
  Rewind file;
  text := ReadStr(file\Delim:=":");
  number := ReadNum(file);
  Close file;
  TPWrite text \Num:=number;
ENDPROC
```

用二进制串行通道通信

此例把字符串“Hello”、机器人当前位置和字符串“Hi”写入了二进制串行通道com1。

```
PROC write_bin_chan()
  VAR iodev channel;
  VAR num out_buffer{20};
  VAR num input;
  VAR rotarget target;

  Open "com1:", channel\Bin;

  ! Write control character enq
  out_buffer{1} := 5;
  WriteBin channel, out_buffer, 1;

  ! Wait for control character ack
  input := ReadBin (channel \Time:= 0.1);
  IF input = 6 THEN
    ! Write "Hello" followed by new line
    WriteStrBin channel, "Hello\0A";

    ! Write current robot position
```



```
target := CRobT(\Tool:= tool1\WObj:= wobj1);
WriteAnyBin channel, target;

! Set start text character (2=start text)
out_buffer{1} := 2;

! Set character "H" (72="H")
out_buffer{2} := 72;
! Set character "i"
out_buffer{3} := StrToByte("i"\Char);
! Set new line character (10=new line)

out_buffer{4} := 10;
! Set end text character (3=end text)

out_buffer{5} := 3;
! Write the buffer with the line "Hi"

! to the channel
WriteBin channel, out_buffer, 5;
ENDIF
Close channel;
ENDPROC
```

2.7.3 原始数据通信

2.7.3.1 概述

目的

原始数据通信的作用是将数据的不同类型打包到一个容器中，然后发送至一份文件或一条串行通道，最后进行数据读取和解压。这在用现场总线（如DeviceNet或Profibus）通信时尤为有用。

其中包括

为了处理原始数据通信，您可通过RobotWare基本功能File and Serial Channel Handling来访问：

- 处理一个rawbytes变量之内容的指令
- 用于读取和写入原始数据的指令
- 用于获取一个rawbytes变量的有效数据长度的函数。

基本方法

这是原始数据通信的一般用法。第84页的写入和读取rawbytes用一个更详细的示例展示了其具体用法。

- 1 将数据打包成一个rawbytes变量（num、byte或string类型的数据）。
- 2 将rawbytes变量写入一份文件或一条串行通道。
- 3 从一份文件或一条串行通道中读取一个rawbytes变量。
- 4 将rawbytes变量解压成num、byte或string。

限制

装置命令通信也需要基本功能Device Command Interface和所论工业网络的选项。不同的RAPID任务不能同时访问各文件、串行通道和现场总线。得通过基于二进制和字符的通信中的所有指令以及WriteRawBytes和ReadRawBytes来进行此类访问。举例来说，如果要在一项任务中执行一条ReadBin指令，那么就必须在另一项任务能执行WriteRawBytes指令前作好准备。

2.7.3.2 RAPID组件

数据类型

此处简述了原始数据通信所用的每种数据类型。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各种数据类型。

数据类型	描述
rawbytes	<p>rawbytes被用作一个通用数据容器，其不但可装入num、byte或string类型的任何数据，还能保存有效数据的长度（以字节计）。</p> <p>rawbytes可容纳多达1024字节的数据。其支持的数据格式有：</p> <ul style="list-style-type: none"> • Hex（1字节） • long（4字节） • float（4字节） • ASCII（1到80个字符）

指令：

此处简述了原始数据通信所用的每条指令。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各条指令。

指令	描述
ClearRawBytes	<p>ClearRawBytes的作用是将一个rawbytes变量的所有内容均设置成0。rawbytes变量中的有效数据长度会被设置成0。</p> <p>也可用ClearRawBytes来仅仅清除一个rawbytes变量的最终部分。</p>
PackRawBytes	PackRawBytes的作用是把类型为num、byte或string的变量的内容打包成一个类型为rawbytes的变量。
UnpackRawBytes	UnpackRawBytes的作用是把类型为rawbytes的变量的内容解压成类型为byte、num或string的变量。
CopyRawBytes	CopyRawBytes的作用是把一个rawbytes变量的全部或部分内容复制到另一个同类变量中。
WriteRawBytes	WriteRawBytes的作用是将类型为rawbytes的数据写入任何二进制文件、串行通道或现场总线。
ReadRawBytes	ReadRawBytes的作用是从任何二进制文件、串行通道或现场总线中读取类型为rawbytes的数据。

函数

此处简述了原始数据通信所用的每则函数。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各则函数。

功能	描述
RawBytesLen	RawBytesLen的作用是获取一个rawbytes变量中的有效数据长度。

2.7.3.3 代码示例

关于示例

这些示例简要示范了如何使用rawbytes。若想通过更真实的示例来了解如何使用DeviceNet通信中的rawbytes，则请参见[第92页的在DeviceNet中写入rawbytes](#)。

写入和读取rawbytes

此例展示了如何将数据打包成一个rawbytes变量以及如何将其写入一部装置中。此外还展示了如何读取和解压一个rawbytes变量。

```
VAR iODEV io_device;
VAR rawbytes raw_data;

PROC write_rawbytes()
  VAR num length := 0.2;
  VAR string length_unit := "meters";

  ! Empty contents of raw_data
  ClearRawBytes raw_data;

  ! Add contents of length as a 4 byte float
  PackRawBytes length, raw_data, (RawBytesLen(raw_data)+1) \Float4;

  ! Add the string length_unit
  PackRawBytes length_unit, raw_data, (RawBytesLen(raw_data)+1)
    \ASCII;

  Open "HOME:" \File:= "FILE1.DOC", io_device \Bin;

  ! Write the contents of raw_data to io_device
  WriteRawBytes io_device, raw_data;

  Close io_device;
ENDPROC

PROC read_rawbytes()
  VAR string answer;

  ! Empty contents of raw_data
  ClearRawBytes raw_data;

  Open "HOME:" \File:= "FILE1.DOC", io_device \Bin;

  ! Read from io_device into raw_data
  ReadRawBytes io_device, raw_data \Time:=1;

  Close io_device;

  ! Unpack raw_data to the string answer
  UnpackRawBytes raw_data, 1, answer \ASCII:=10;
```

下一页继续

```
ENDPROC
```

复制rawbytes

在此例中，raw_data_1和raw_data_2的所有数据都被复制到了raw_data_3。

```
VAR rawbytes raw_data_1;
VAR rawbytes raw_data_2;
VAR rawbytes raw_data_3;
VAR num my_length:=0.2;
VAR string my_unit:=" meters";

PackRawBytes my_length, raw_data_1, 1 \Float4;
PackRawBytes my_unit, raw_data_2, 1 \ASCII;

! Copy all data from raw_data_1 to raw_data_3
CopyRawBytes raw_data_1, 1, raw_data_3, 1;

! Append all data from raw_data_2 to raw_data_3
CopyRawBytes raw_data_2, 1, raw_data_3, (RawBytesLen(raw_data_3)+1);
```

2.7.4 文件与目录管理

2.7.4.1 概述

目的

文件与目录管理的目的是便于浏览和编辑文件结构（目录和文件）。

其中包括

为了处理文件与目录管理事项，您可通过RobotWare基本功能File and Serial Channel Handling来访问：

- 处理目录的指令
- 用于读取目录的一则函数
- 在某文件结构等级上处理文件的指令
- 检索大小信息和类型信息的函数。

基本方法

这是文件与目录管理的一般方式。[第88页的代码示例](#)用一个更详细的示例展示了其具体做法。

- 1 打开一个目录。
- 2 从该目录中进行读取，并一直搜索到发现您的搜索对象为止。
- 3 关闭该目录。

2.7.4.2 RAPID组件

数据类型

此处简述了文件与目录管理所用的每种数据类型。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各种数据类型。

数据类型	描述
dir	dir包含了硬盘或网络上的某目录的引用项。可用指令OpenDir将其与物理目录链接起来。

指令：

此处简述了文件与目录管理所用的每条指令。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各条指令。

指令	描述
OpenDir	OpenDir的作用是打开一个目录。
CloseDir	CloseDir的作用是关闭一个目录。
MakeDir	MakeDir的作用是创建一个新目录。
RemoveDir	RemoveDir的作用是移除一个空目录。
CopyFile	CopyFile的作用是复制一份现有文件。
RenameFile	RenameFile的作用是为一份现有文件提供一个新的名称，此外也可在该目录结构下将一份文件从一处移到另一处。
RemoveFile	RemoveFile的作用是移除一份文件。

函数

此处简述了文件与目录管理所用的每则函数。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各则函数。

功能	描述
ReadDir	ReadDir的作用，是在已用指令OpenDir打开的一个目录中检索下一份文件或子目录的名称。 请注意，ReadDir读取的第一批条目是.（全停符）..（双重全停符），即当前目录及其上层目录的符号表达。
FileSize	FileSize的作用是检索指定文件的大小（以字节计）。
FSSize	FSSize（文件系统大小）的作用是检索指定文件所在文件系统的大小（以字节计）。FSSize可检索该系统的总大小或自由大小。
IsFile	IsFile的作用是测试指定文件的类型是否为指定类型，同时也可用于测试相关文件是否存在。

2.7.4.3 代码示例

列出文件

此例展示了如何列出一个目录中的相关文件，但该目录本身及其上层目录不在此列（和..）。

```
PROC lsdir(string dirname)
  VAR dir directory;
  VAR string filename;

  ! Check that dirname really is a directory
  IF IsFile(dirname \Directory) THEN
    ! Open the directory
    OpenDir directory, dirname;

    ! Loop through the files in the directory
    WHILE ReadDir(directory, filename) DO
      IF (filename <> "." AND filename <> ".." THEN
        TPWrite filename;
      ENDIF
    ENDWHILE

    ! Close the directory
    CloseDir directory;
  ENDIF
ENDPROC
```

将文件移至新目录

此例创建了一个新文件，重命名了一份文件并将其移至新目录，此外还移除了旧目录。

```
VAR dir directory;
VAR string filename;

! Create the directory newdir
MakeDir "HOME:/newdir";

! Rename and move the file
RenameFile "HOME:/olddir/myfile", "HOME:/newdir/yourfile";

! Remove all files in olddir
OpenDir directory, "HOME:/olddir";
WHILE ReadDir(directory, filename) DO
  IF (filename <> "." AND filename <> ".." THEN
    RemoveFile "HOME:/olddir/" + filename;
  ENDIF
ENDWHILE
CloseDir directory;

! Remove the directory olddir (which must be empty)
RemoveDir "HOME:/olddir";
```

检查大小

此例将相关文件的大小与文件系统中的剩余自由空间作了对比。若还有足够空间，系统便会复制该文件。

```
VAR num freefsyssize;  
VAR num f_size;  
  
! Get the size of the file  
f_size := FileSize("HOME:/myfile");  
  
! Get the free size on the file system  
freefsyssize := FSSize("HOME:/myfile" \Free);  
  
! Copy file if enough space free  
IF f_size < freefsyssize THEN  
    CopyFile "HOME:/myfile", "HOME:/yourfile";  
ENDIF
```

2.8 Device Command Interface

2.8.1 Device Command Interface介绍

目的

Device Command Interface提供了一个与工业网络上的I / O装置进行通信的接口。请配合原始数据通信来使用该接口，具体请参见[第82页的原始数据通信](#)。

其中包括

您可通过RobotWare基本功能Device Command Interface来访问：

- 用于创建一个DeviceNet标题的指令。

基本方法

这是Device Command Interface的一般用法。[第92页的在DeviceNet中写入rawbytes](#)用一个更详细的示例展示了其具体用法。

- 1 在一个rawbytes变量中添加一个DeviceNet标题。
- 2 在rawbytes变量中添加相关数据。
- 3 向DeviceNet I / O写入rawbytes变量。
- 4 从DeviceNet I / O的数据中读取一个rawbytes变量。
- 5 从rawbytes变量中提取相关数据。

限制

装置命令通信也需要基本功能File and Serial Channel Handling和所论工业网络的选项。

以下类型的工业网络支持Device Command Interface：

- DeviceNet
- EtherNet/IP

2.8.2 RAPID部件和系统参数

数据类型

没有针对Device Command Interface的RAPID数据类型。

指令：

此处简述了Device Command Interface中的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
PackDNHeader	PackDNHeader在一个rawbytes变量中添加了一个DeviceNet标题，而该标题则指定了将开展的一种服务（如设置或获取）和DeviceNet I / O装置上的一个参数。

函数

没有Device Command Interface所需的RAPID函数。

系统参数

没有Device Command Interface中的特定系统参数。至于系统函数方面的一般信息，则请参见技术参考手册 - 系统参数。

2.8.3 代码示例

在DeviceNet中写入rawbytes

此例把已打包的数据作为一个rawbytes变量写入了一部DeviceNet I/O装置。

rawbytes方面的更多信息请参见[第82页的原始数据通信](#)。

```
PROC set_filter_value()
  VAR iodev dev;
  VAR rawbytes rawdata_out;
  VAR rawbytes rawdata_in;
  VAR num input_int;
  VAR byte return_status;
  VAR byte return_info;
  VAR byte return_errcode;
  VAR byte return_errcode2;

  ! Empty contents of rawdata_out and rawdata_in
  ClearRawBytes rawdata_out;
  ClearRawBytes rawdata_in;

  ! Add DeviceNet header to rawdata_out with service
    "SET_ATTRIBUTE_SINGLE" and path to filter attribute on
    DeviceNet I/O device
  PackDNHeader "10", "6,20 1D 24 01 30 64,8,1", rawdata_out;

  ! Add filter value to send to DeviceNet I/O device
  input_int:= 5;
  PackRawBytes input_int, rawdata_out, (RawBytesLen(rawdata_out) +
    1) \IntX := USINT;

  ! Open I/O device
  Open "/FCI1:" \File:="board328", dev \Bin;

  ! Write the contents of rawdata_out to the I/O device
  WriteRawBytes dev, rawdata_out \NoOfBytes :=
    RawBytesLen(rawdata_out);

  ! Read the answer from the I/O device
  ReadRawBytes dev, rawdata_in;

  ! Close the I/O device
  Close dev;

  ! Unpack rawdata_in to the variable return_status
  UnpackRawBytes rawdata_in, 1, return_status \Hex1;

  IF return_status = 144 THEN
    TPWrite "Status OK from device. Status code:
      "\Num:=return_status;
  ELSE
    ! Unpack error codes from device answer
```

```
UnpackRawBytes rawdata_in, 2, return_errcode \Hex1;  
UnpackRawBytes rawdata_in, 3, return_errcode2 \Hex1;  
TPWrite "Error code from device: " \Num:=return_errcode;  
TPWrite "Additional error code from device: "  
    \Num:=return_errcode2;  
ENDIF  
ENDPROC
```

2.9 Logical Cross Connections

2.9.1 Logical Cross Connections介绍

目的

Logical Cross Connections的作用是检查和影响各数字I / O信号（DO、DI）或编组I / O信号（GO、GI）的组合。可由此验证或控制相关机器人之外的工艺设备。此项功能相当于一个简单的PLC。

若令I / O系统用I / O信号处理逻辑运算，则可避免执行许多RAPID代码。Logical Cross Connections可取代进程“读取I / O信号值、计算新值，并在I / O信号中写入数值”。

此处是一些应用示例：

- 当三个输入信号中的任意一个被设置成1时，系统便会中断程序执行过程。
- 若两个输入信号都被设置成1，那么就把一个输出信号设置成1。

描述

Logical Cross Connections的作用是定义一个I / O信号与其它I / O信号之间的依赖性。可用逻辑运算符AND和OR以及反信号值来配置更为复杂的依赖性。

若I / O信号由相应的逻辑表达式（执行I / O信号）和该表达式所得I / O信号（合成I / O信号）构成，那么该信号就可以是数字I / O信号（DO、DI）或编组I / O信号（GO、GI）。

其中包括

通过Logical Cross Connections，您最多可用5个执行I / O信号、逻辑运算AND和OR以及反信号值来构建逻辑表达式。

2.9.2 配置Logical Cross Connections

系统参数

此处简述了交叉连接所用的相关参数。更多信息请参见第95页的配置*Logical Cross Connections*中的各个参数。

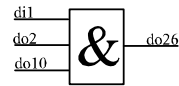
这些参数属于主题*I/O System*下的类型*Cross Connection*。

参数	描述
<i>Name</i>	指定该交叉连接的名称。
<i>Resultant</i>	把交叉连接的结果作为自身新值来接收的I / O信号。
<i>Actor 1</i>	求取 <i>Resultant</i> 时将使用的第一个I / O信号。
<i>Invert actor 1</i>	如果 <i>Invert actor 1</i> 被设置成Yes, 那么求取 <i>Resultant</i> 时就使用 <i>Actor 1</i> 的反值。
<i>Operator 1</i>	<p><i>Actor 1</i>与<i>Actor 2</i>之间的运算元。</p> <p>可以是这些运算元之一：</p> <ul style="list-style-type: none"> AND——如果两个输入值都为1, 则得出该值为1。 OR——如果至少有一个输入值为1, 则得出该值为1。 <p> 注意</p> <p>从左到右计算相关的运算符（即从<i>Operator 1</i>开始, 到<i>Operator 4</i>结束）。</p>
<i>Actor 2</i>	求取 <i>Resultant</i> 时将使用的第二个I / O信号（若此类信号超过一个）。
<i>Invert actor 2</i>	如果 <i>Invert actor 2</i> 被设置成Yes, 那么求取 <i>Resultant</i> 时就使用 <i>Actor 2</i> 的反值。
<i>Operator 2</i>	<p><i>Actor 2</i>与<i>Actor 3</i>之间的运算元。</p> <p>参见<i>Operator 1</i>。</p>
<i>Actor 3</i>	求取 <i>Resultant</i> 时将使用的第三个I / O信号（若此类信号超过两个）。
<i>Invert actor 3</i>	如果 <i>Invert actor 3</i> 被设置成Yes, 那么求取 <i>Resultant</i> 时就使用 <i>Actor 3</i> 的反值。
<i>Operator 3</i>	<p><i>Actor 3</i>与<i>Actor 4</i>之间的运算元。</p> <p>参见<i>Operator 1</i>。</p>
<i>Actor 4</i>	求取 <i>Resultant</i> 时将使用的第四个I / O信号（若此类信号超过三个）。
<i>Invert actor 4</i>	如果 <i>Invert actor 4</i> 被设置成Yes, 那么求取 <i>Resultant</i> 时就使用 <i>Actor 4</i> 的反值。
<i>Operator 4</i>	<p><i>Actor 4</i>与<i>Actor 5</i>之间的运算元。</p> <p>参见<i>Operator 1</i>。</p>
<i>Actor 5</i>	求取 <i>Resultant</i> 时将使用的第五个I / O信号（若此类信号超过四个）。
<i>Invert actor 5</i>	如果 <i>Invert actor 5</i> 被设置成Yes, 那么求取 <i>Resultant</i> 时就使用 <i>Actor 5</i> 的反值。

2.9.3 示例

逻辑AND

以下逻辑结构.....



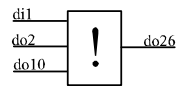
xx0300000457

.....创建如下。

Resultant	Actor 1	Invert actor 1	Operator 1	Actor 2	Invert actor 2	Operator 2	Actor 3	Invert actor 3
do26	di1	No	AND	do2	No	AND	do10	No

逻辑OR

以下逻辑结构.....



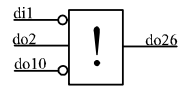
xx0300000459

.....创建如下。

Resultant	Actor 1	Invert actor 1	Operator 1	Actor 2	Invert actor 2	Operator 2	Actor 3	Invert actor 3
do26	di1	No	OR	do2	No	OR	do10	No

反信号

以下逻辑结构（一个环形代表一个反信号）



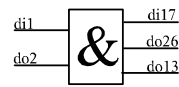
xx0300000460

.....创建如下。

Resultant	Actor 1	Invert actor 1	Operator 1	Actor 2	Invert actor 2	Operator 2	Actor 3	Invert actor 3
do26	di1	Yes	OR	do2	No	OR	do10	Yes

若干结果

无法用一条交叉连接来执行以下逻辑结构.....



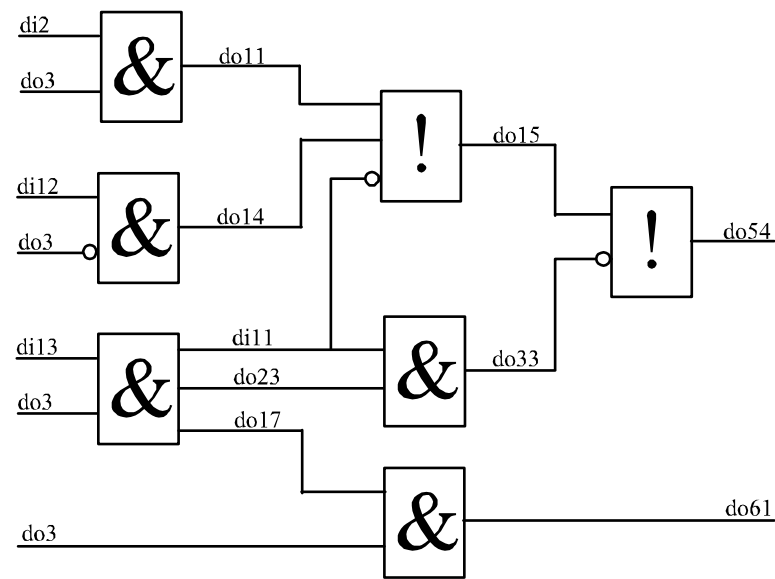
xx0300000462

.....但若有三条交叉连接， 则能执行如下。

Resultant	Actor 1	Invert actor 1	Operator 1	Actor 2	Invert actor 2
di17	di1	No	AND	do2	No
do26	di1	No	AND	do2	No
do13	di1	No	AND	do2	No

复杂条件

以下逻辑结构.....



xx0300000461

.....创建如下。

Resultant	Actor 1	Invert actor 1	Operator 1	Actor 2	Invert actor 2	Operator 2	Actor 3	Invert actor 3
do11	di2	No	AND	do3	No			
do14	di12	No	AND	do3	Yes			
di11	di13	No	AND	do3	No			
do23	di13	No	AND	do3	No			
do17	di13	No	AND	do3	No			
do15	do11	No	OR	do14	No	OR	di11	Yes
do33	di11	No	AND	do23	No			
do61	do17	No	AND	do3	No			
do54	do15	No	OR	do33	Yes			

2.9.4 限制

求值顺序

如果在一条交叉连接中使用了两个以上的执行I / O信号，那么从左到右依次求值。这意味着先对Actor 1与Actor 2之间的运算进行求值，然后将其求取结果用在与Actor 3有关的运算中。

如果一条交叉连接中的所有运算符均属同一类型（仅为AND或仅为OR），那么求值顺序没什么影响。不过若在不考虑求值顺序的情况下混用AND和OR运算符，则可能产生意料之外的结果。



提示

采用若干条交叉连接，而不是在同一条交叉连接中混用AND和OR。

执行I / O信号的最大数目

一提案交叉连接的执行I / O信号可能不会超过五个。若需更多执行I / O信号，则请采用多条交叉连接。

交叉连接的最大次数

本机器人系统最多处理300条交叉连接。

最大深度

交叉连接求值的最大允许深度为20。

可将一条交叉连接的结果作为另一条交叉连接的执行器，而后的结果则可依序作为下一条交叉连接的执行器。不过这种依赖性交叉连接链的深度不能超过20步。

勿创建环路

交叉连接不得出自闭合链，否则会导致无限次的求值和振荡。闭合链会出现在各交叉连接相互关联之时，并使交叉连接链形成一个循环。

相同的结果勿出现一次以上

不得使用模棱两可的合成I / O信号，否则相关结果将取决于求值顺序（无法控制这种顺序）。当若干条交叉连接均得出同一I / O信号后时，便会出现模棱两可的合成I / O信号。

重叠各装备映射

对一条交叉连接中的合成I / O信号而言，其装置映射不得与该交叉连接所定义的任何反向执行I / O信号重叠。使用交叉连接中存在重叠装置映射的I / O信号会导致无限次信号设定环路。

2.10 Remote Service Embedded

2.10.1 概述

描述

Remote Service是对连接到云端 的ABB Remote Service Centre的ABB robot controllers的可用功能。

此前Remote Service功能曾部署在连接到控制器服务端口的外部硬件(Remote Service Box)上。Remote Service Box提供了服务数据集合以及外部连接方式（无线GPRS、3G或有线网络）。

Remote Service Embedded或 RSE是RobotWare内远程服务的软件版本。

目的

Remote Service Embedded的主要目的是在机器人控制器由用户通过其WAN端口连接到互联网时，消除使用外部硬件的需要。

Remote Service Embedded在RobotWare中原生可用，插入即可连接到：

- 为控制器提供连接。
- 启用并注册连接的控制器到远程服务。

ABB 3G/4G/Wifi网关将在未来提供，以便使用无线连接。

其中包括

RobotWare基础功能Remote Service Embedded为您提供：

- RSE代理程序，管理连接和服务数据集合。
- 系统参数，用于启用RSE和配置连接。
- 远程服务的关键事件的专用事件日志。
- 系统信息中可用的状态和信息页。

操作前提

Remote Service功能要求在《远程服务协议》中定义控制器。请 联系当地ABB服务部门创建《远程服务协议》并获取对MyRobot网站的访问，以便在连接后进行注册。



注意

MyRobot是在《远程服务协议》下提供机器人控制器服务信息的ABB网站。

基本流程

以下为建立Remote Service Embedded的基本流程。

- 1 配置控制器的互联网连接。
- 2 启用Remote Service Embedded并启动连接。
- 3 通过MyRobot注册页面注册控制器。

下一页继续

RSE连接并注册后，服务数据集合将会在后台透明运行。



注意

使用**System Info Remote Service**页面可了解信息和进行本地注册。

使用**MyRobot**网站可了解全部远程服务功能以及进行远程侧注册。

限制

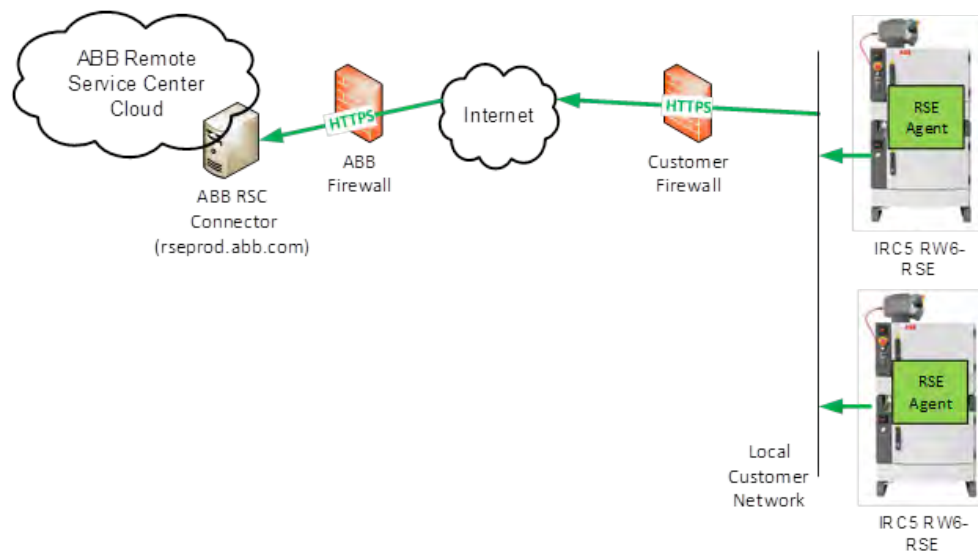
以下为Remote Service Embedded的限制：

- 控制器的识别是通过控制器序列号完成的，且必须与《远程服务协议》中确定的序列号匹配。
- 客户还必须为机器人控制器提供到公共互联网的连接，或使用ABB无线网关（如有提供）。

2.10.2 RSE连接

RSE连接概念

Remote Service Embedded的概念是在控制器内部实现一个虚拟的RSE代理，由代理通过互联网与ABB远程服务中心进行安全的通信。通信采用HTTPS（安全HTTP）方式，并仅限从控制器连接到ABB RSC连接器，以确保客户网络与任何外部互联网访问隔离。下图介绍了这些概念：



xx1500003224

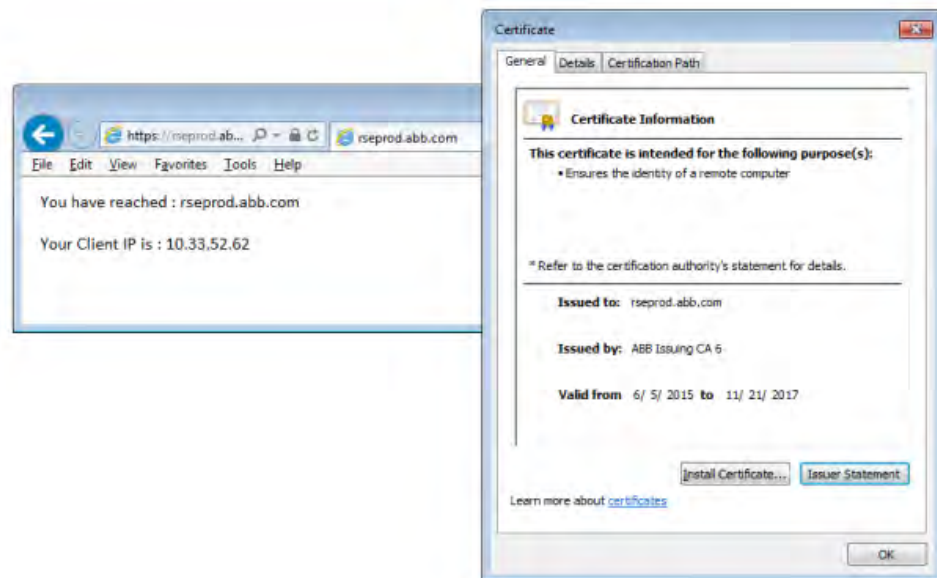
2 RobotWare-OS

2.10.2 RSE连接

续前页

故障排除

您可以从所在位置验证控制器到RSE服务器的连接情况。这可以通过用同样的网络配置（WAN IP/子网掩码、DNS、路由）来连接一台PC（而非控制器），并在浏览器中打开服务器的根路径(<https://rseprod.abb.com>)。如果DNS名称解析成功，则验证通过，浏览器会显示一个页面，显示RSE服务器信息，并使用一个ABB证书来加密，如下图所示。



xx1500003225

2.10.3 配置 - 系统参数

简介

本节简要介绍了用于Remote Service Embedded的系统参数。如需了解更多信息，请参阅技术参考手册 - 系统参数 - *Remote Service Connection*。

远程服务连接

下列参数属于主题*Communication*和类型*Remote Service Connection*。如需了解更多信息，请参阅技术参考手册 - 系统参数 - *Remote Service Connection*中的相关参数。

参数	描述
Enabled	启用或禁用RSE。如果禁用RSE，则不会有任何来自控制器的通信。
Connection Type	表明通信是通过客户网络还是适用ABB移动网关解决方案（将在未来产品中提供）完成的。
Connection Cost	根据可用连接的类型调整轮询速度和流量： <ul style="list-style-type: none"> 命令轮询（低）1分钟，（中）10分钟，（高）1小时。 注册轮询（低）10分钟，（中）30分钟，（高）2小时。
Proxy Used, Name, Port	表明是否要求代理来访问互联网及代理名称与端口。
Gateway IP Address	ABB移动网关解决方案的IP地址（将在未来的产品中提供）。

WAN配置

WAN IP/掩码/网关配置在启动应用、设置中完成。提供互联网访问的WAN以太网端口配置需要在控制器上进行。端口由其IP、掩码以及可能的网关确定。有关WAN配置的详细信息，请参阅应用手册 - *EtherNet/IP Scanner/Adapter*中的硬件概述

DNS配置

这些参数属于*Communication*主题和*DNS Client*类型。如果未使用ABB移动网关，要将ABB RSE连接器(rseprod.abb.com)的名称解析到其IP地址，则需要定义DNS服务器。详情请参阅技术参考手册 - 系统参数中的*Type DNS Client*。

IP路由配置

这些参数属于*Communication*主题和*IP Routing*类型。在某些情况下，需要定义一些路由参数来表明在客户网络中哪个外部设备作为访问互联网的网关。默认情况下，IP路由是以WAN端口上定义的网关为基础创建的。但如果不应该使用默认路由，则可以添加一个特定路由。详情请参阅技术参考手册 - 系统参数中的*Type IP Route*。

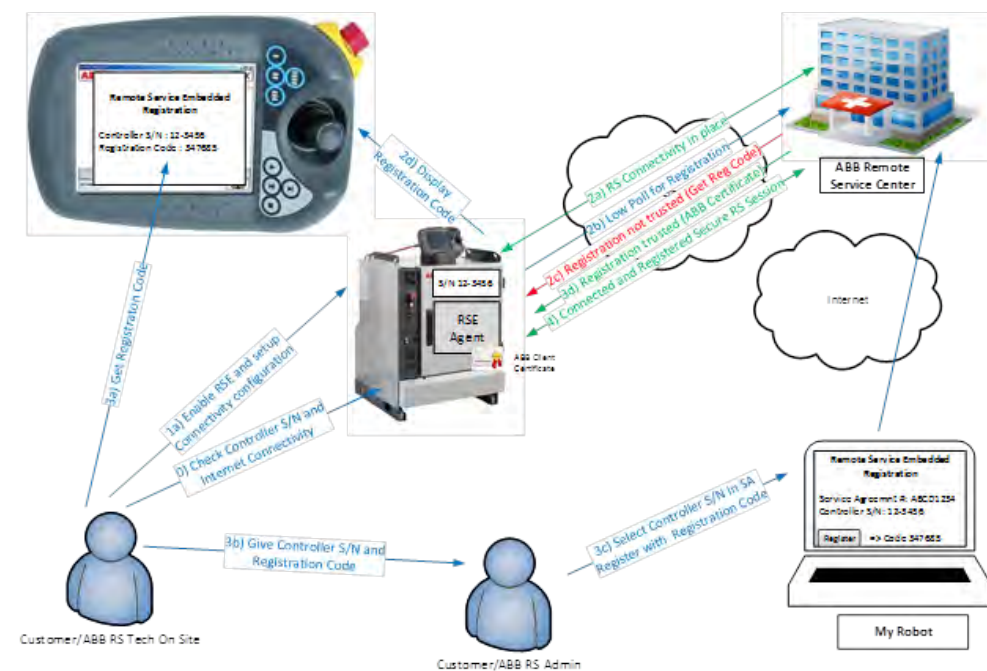
2.10.4 RSE注册

RSE启动

RSE启动以下列步骤为基础：

- (0) RSE准备
- (1) RSE配置
- (2) RSE连接
- (3) RSE注册
- (4) RSE已连接和注册

当这些步骤完成后，RSE代理会使用机器人控制器中安装的ABB证书安全连接并识别，类似外部远程服务箱。下图介绍了这些概念：



RSE准备

- 用控制器模块柜中找到的序列号验证控制器序列号。
- 验证并为机器人控制器提供互联网连接。
- 验证此控制器的服务协议是否随ABB机器人服务提供。

RSE配置

- 配置连接参数
- 启用远程服务

RSE连接

- RSE代理连接到ABB远程服务中心。
- 初始注册过程以低轮询速率开始。
- 初始注册不完整，尚未完全信任。

下一页继续

- 已收到注册码来完成信任关系。
- 注册码在远程服务注册页上提供。

RSE注册

- 客户/ABB现场提供控制器序列号和注册码给远程服务管理员进行注册。
- 远程服务管理员在其服务协议中的MyRobot部分验证注册码。
- 注册信任开始，在控制器部署一个客户证书。

RSE连接和注册完成

- 控制器已连接和注册，并在服务协议中确认。
- 连接使用ABB证书信任连接。
- 远程服务现在正在机器人控制器上运行中。

2.10.5 远程服务信息

远程服务页面

简介

RSE信息页在系统信息 > 软件资源 > 通信 > 远程服务中提供。下列为4个RSE信息页：

- 概述
- 服务器连接
- 注册
- **Advanced (高级)**



注意

页面显示的信息可以通过换页或按刷新按钮来刷新。如果RSE代理在等待中，刷新按钮也可以强制建议与服务器的连接。（例如，等待MyRobot的注册确认）。这在当连接成本设置为High时采用低轮询时适用。

概述页

概述页提供了RSE状态和信息的概况。如果状态并非活动，则其他页提供更详细的信息。

字段	描述	可能值	示例
启用	显示开关RSE的主配置开关值。	Yes/No	Yes
状态	显示当前状态以查看是否有需要浏览到服务器连接页或注册页。	"-" Failed Initializing Shutdown Registration in progress Trying to connect Active	Active
序列号	显示用于在远程服务中识别控制器的标识符。	Controller Serial number	12-45678
RobotWare版本	显示发送到服务器的RobotWare版本。	RobotWare version name	6.03.0088
重启计数器	显示RSE代理自动重启的次数。这可用于确认看门狗程序是否已经重启了RSE代理。	0-N If not Enabled, then display: 0	2
脚本版本	显示下载的数据采集器代码版本。	"数据采集器脚本名称" "-"	0116/ROBOTWARE-6.02.0000+/5196
服务协议	用于确认控制器是否与预期的服务协议关联。	"服务协议名称" "-"	SA_FR12_16
客户名称	用于确认控制器是否与预期的服务协议关联。	"服务协议的客户名称" "-"	ABB Robotics
国家/地区	用于确认控制器是否与预期的服务协议关联。	"服务协议的国家/地区" "-"	法国

下一页继续

字段	描述	可能值	示例
刷新按钮	刷新时，RSE代理会使用当前数据响应并中断等待状态（如果在等待中），以联系服务器并刷新信息。		

服务器连接页

服务器连接页提供了RSE到服务器连接的概况。

字段	描述	可能值	示例
状态	显示当前状态以查看是否有需要浏览到服务器连接页或注册页。	"-" Failed Initializing Shutdown Registration in progress Trying to connect Active	Active
连接状态	显示与服务器通信的状态以及错误类型。	Initializing Server not reachable Server not authenticated Server error (HTTP xxxx) Connected	Connected
上次更新	显示自从服务器连接页生成以来经过的相对时间。		“HH:MM:SS前”
服务器名称	显示配置了RSE代理的服务器的名称。	"" 服务器名称	rseprod.abb.com
服务器IP	显示服务器用于连接的IP地址和端口号。IP地址是RSE代理进行DNS名称解析得到的。	"" 服务器IP	138.227.175.43
服务器证书名称	显示证书名称信息。	"" 服务器名称 不受信任（服务器）	rseprod.abb.com
服务器证书签发方	显示证书签发方的名称。	"" 签发方 不受信任（签发方）	ABB签发CA 6
服务器证书有效期至	显示证书日期。	"" 签发方 失效（日期）	Nov 21 07:09:28 2017 GMT
控制器时间	显示控制器日期和时间详情。		16-01-08 13:52:33
DNS服务器	显示DNS信息。	不可用 DNS值	10.0.23.45
刷新按钮	刷新时，RSE代理会使用当前数据响应并中断等待状态（如果在等待中），以联系服务器并刷新信息。		

下一页继续

2 RobotWare-OS

2.10.5 远程服务信息

续前页

注册页

注册页面提供了RSE注册的概况。

字段	描述	可能值	示例
状态	显示当前状态以查看是否有需要浏览到服务器连接页或注册页。	"-" Failed Initializing Shutdown Registration in progress Trying to connect Active	活动
注册状态	显示注册状态与注册码。	Register with code in MyRobot Registration in progress Registered Failed	在MyRobot中用注册码注册
注册码	显示注册码。此代码用于登录到MyRobot。	"-" 代码值	456735
刷新按钮	刷新时，RSE代理会使用当前数据响应并中断等待状态（如果在等待中），以联系服务器并刷新信息。		

高级页

高级页面提供了有关RSE代理与服务器之间对话的高级信息。

字段	描述	可能值	示例
上一个HTTP消息	显示发出的上一个消息。	Register CheckRegister GetLoginInfo GetMessage ...	GetMessage
最后HTTP日期	显示上一个上个消息发送时的日期和时间。		发送于HH:MM:SS前
上一个HTTP错误	显示上一个消息发送时如果消息ID为4XX的HTTP错误。	Not Available 错误HTTP XXX + 消息	不可用
下一个消息	显示要发送的下一个消息以及发送消息的日期。		70秒内获取消息
上一个命令	显示从服务器接收的上一个命令。	Not Available Reboot Reset Ping Diagnostic ...	不可用
刷新按钮	刷新时，RSE代理会使用当前数据响应并中断等待状态（如果在等待中），以联系服务器并刷新信息。		

下一页继续

远程服务日志


RSE代理在中心控制器事件日志中生成部分事件日志。控制器在启动、注册、取消注册、失去连接以及其他重要事件期间生成。

事件日志在170XXX的范围内，与其他控制器事件日志文档一起介绍。详情请参阅操作员手册 - IRC5 故障排除。

强制RSE代理的重置

RSE代理是可以重置的。重置时，RSE代理会清除其全部内部信息，包括注册信息、数据采集器脚本和全部本地存储的服务信息。配置将不会重置，但是需要重新注册来重新激活远程服务。

使用下述步骤重置RSE代理：

	操作
1	点击ABB按钮来显示ABB菜单。 菜单中列出处理程序。
2	点击程序编辑器 -> Debug -> 调用。  注意 如果Debug禁用，点击PP到Main。
3	点击RemoteServiceReset -> 前往。在控制器上按电机上电按钮。
4	按播放按钮来执行重置例行程序 -> 点击重置。

此页刻意留白

3 Motion performance

3.1 Absolute Accuracy [603-1, 603-2]

3.1.1 关于Absolute Accuracy

目的

Absolute Accuracy是一种校准概念，此概念确保了整个工作范围内的TCP准确度在大多数情况下都优于 ± 1 毫米。

出于机械容限和机器人结构的偏移之故，理想机器人与真实机器人之间可能相差数个毫米。Absolute Accuracy会补偿这些差值，从而确保给定坐标符合实际的机器人位置。

这里有一些示例说明了这种准确度在何时意义重大：

- 最低程度修整下的离线编程。
- 机器人的可交换性。
- 用工具来准确重定方位的离线编程
- 重新使用各应用之间的程序

包括哪些

通过下列项目来交付每台Absolute Accuracy机器人：

- 该机器人系列测量板上保存的补偿参数
- 一份出厂证书，代表了校准与验证序列所用的Absolute Accuracy测量协议。

辨认一台Absolute Accuracy机器人

具有Absolute Accuracy校准的机器人会在机械臂（靠近铭牌处）上标注一个如下符号：



xx0300000314

基本方法

这是在您的机器人上设置Absolute Accuracy的基本步骤，更详细的信息请参见[第114页的激活Absolute Accuracy](#)。

- 1 激活Absolute Accuracy。
- 2 重启控制器。

限制

Absolute Accuracy的作用对象是笛卡儿坐标上的一个机器人目标点，而并非单个关节，因此基于关节的移动（如MoveAbsJ）将不受影响。具体请参见[第112页的何时使用Absolute Accuracy](#)。

如果机器人被悬挂起来，那么就必须在悬挂机器人时进行Absolute Accuracy校准。相关补偿参数将取决于机器人是安装在地面还是被悬挂起来。

3 Motion performance

3.1.2 何时使用Absolute Accuracy

3.1.2 何时使用Absolute Accuracy

概述

当Absolute Accuracy被激活时，机器人仍会使用如常，但Absolute Accuracy处于激活状态。不过Absolute Accuracy仅在涉及笛卡尔坐标（即机器人目标点）时才会起到作用。Absolute Accuracy不会影响到基于关节的移动（即关节目标点）。

就Absolute Accuracy的激活状态而言，下表定义了其会在何时处于激活状态。为了作进一步的解释，这里又用一些示例说明了其在何时不会处于激活状态。

Absolute Accuracy处于激活状态

下列情况将会激活Absolute Accuracy：

- 机器人目标点上有任何基于函数的运动（MoveL），或对机器人目标点进行了ModPos
- 重定方位点动
- 线性点动
- 工具定义（4、5和6点工具定义，空间固定TCP，固定工具）
- 工件定义

Absolute Accuracy未处于激活状态

以下示例说明了Absolute Accuracy何时不会处于激活状态：

- 关节目标点上任何基于函数的运动（MoveAbsJ）
- 独立关节
- 基于关节的点动
- 附加轴
- 动作跟踪

3.1.3 有用工具

概述

运行和维护Absolute Accurate机器人时建议使用以下产品：

- Load Identification
- 校准单摆（标准的机器人校准工具）
- CalibWare（Absolute Accuracy校准工具）

Load Identification

Absolute Accuracy根据有效负载计算了机器人的偏移。准确描述相关负载可以说非常重要。

Load Identification是一种工具，其作用是确定有效负载的质量、重心和惯量。

有关更多信息，请参阅 操作员手册 - 带 *FlexPendant* 的 *IRC5*。

Calibration Pendulum

Calibration Pendulum的作用是校准机器人的旋转变压器偏移。这意味着机器人会处在其起始位置（所有轴的角度均被设置成零），并对旋转变压器进行校准。

有关方面为不同机器人型号推荐了不同的旋转变压器偏移校准工具，其中最常用的工具是Calibration Pendulum。用户可在各台机器人的产品手册和操作员手册 - *Calibration Pendulum*中找到该机器人的校准信息。

在首次校准和保养机器人时使用Calibration Pendulum。

CalibWare

ABB公司提供了CalibWare作为校准Absolute Accuracy的工具。CalibWare的文件详细描述了Absolute Accuracy无返回值校准程序。

在首次校准和保养机器人时使用CalibWare。

3 Motion performance

3.1.4 配置

3.1.4 配置

激活Absolute Accuracy

使用RobotStudio，然后遵守下列步骤（更多信息请参见操作员手册 - *RobotStudio*）：

	操作
1	如果您尚无写入权限，则点击请求写入权限，然后等待FlexPendant示教器的授权。
2	点击配置编辑器，然后选择运动。
3	点击类型机器人。
4	配置参数 <i>Use Robot Calibration</i> ，然后将数值更改为“r1_calib”。
5	至于MultiMove系统，则在每台机器人上重复步骤3和4，然后将机器人2、机器人3和机器人4的 <i>Use Robot Calibration</i> 分别设置成“r2_calib”、“r3_calib”和“r4_calib”。
6	重启控制器，从而使所做改动生效。



提示

若要验证Absolute Accuracy是否处于激活状态，则请查看FlexPendant上的点动窗口。当Absolute Accuracy处于激活状态时，左侧窗口会显示“Absolute Accuracy On”字样。对于MultiMove系统，则请检查所有机械单元的这一状态。

停用Absolute Accuracy

使用RobotStudio，然后遵守下列步骤（更多信息请参见操作员手册 - *RobotStudio*）：

	操作
1	如果您尚无写入权限，则点击请求写入权限，然后等待FlexPendant示教器的授权。
2	点击配置编辑器，然后选择主题运动。
3	点击类型机器人。
4	配置参数 <i>Use Robot Calibration</i> ，然后将数值更改为“r1_uncalib”。
5	至于MultiMove系统，则在每台机器人上重复步骤3和4，然后将机器人2、机器人3和机器人4的使用机器人校准分别设置成“r2_uncalib”、“r3_uncalib”和“r4_uncalib”。
6	重启控制器，从而使所做改动生效。

更改校准数据

若您交换了机械臂，则必须载入新机械臂的calibration数据，也就是把该机器人系统测量板中的校准数据复制到相应的机器人控制器上。

使用FlexPendant示教器，然后遵守下列步骤（更多信息请参见操作员手册 - 带*FlexPendant*的*IRC5*）：

	操作
1	轻击ABB菜单，然后是校准。
2	轻击想要更新的机器人。
3	点击机器人内存选项卡。
4	轻击高级。
5	点击清除控制器内存。
6	轻击清除，然后通过轻击是加以确认。

下一页继续

	操作
7	轻击关闭。
8	点击 Update（更新）。
9	点击机柜或机器人已交换并点是确认。

3 Motion performance

3.1.5.1 影响准确度的维护

3.1.5 维护

3.1.5.1 影响准确度的维护

概述

本节将关注那些会直接影响到机器人准确度的维护活动，并总结如下：

- 工具的重新校准
- 电机的更换
- 腕的更换（大型机器人）
- 臂的更换（低臂、高臂、齿轮箱和足）
- 机械臂的更换
- 丧失准确度

工具的重新校准

重新校准工具方面的更多信息请参见第132页的工具校准。

电机的更换

若更换了小型机器人上的所有电机或更换了大型机器人上的轴1到轴4（比如IRB 6700），则需用Calibration Pendulum重新校准相应的旋转变压器偏移参数。

各台机器人的产品手册描述了这一校准进程。

腕的更换

若更换了大型机器人上的腕单元（比如IRB 6700），则需用Calibration Pendulum重新校准轴5和轴6的旋转变压器偏移。


各台机器人的产品手册描述了这一校准进程。

臂的更换

更换任何机器人臂或其它机械结构（腕除外）都会使机器人的结构发生显著变化，因而需要重新校准机器人。建议更换臂后最好重新校准整台机器人，宜确保Absolute Accuracy功能处于最佳状态。重新校准时通常采用CalibWare和一套单独的测量系统。CalibWare可与任何通用的3D测量系统搭配使用。

至于此校准进程方面的更多信息，则请参见CalibWare的文件。

此校准进程总结如下：

	操作
1	更换受影响的部件。
2	对所有轴进行旋转变压器校准。参见各台机器人的产品手册。
3	重新校准TCP。
4	通过比较围笼中的固定参考点来检查相应的准确度。
5	检查相关工件的准确度。  注意 更新已定义的工件将会减少定位时的偏差。

下一页继续

	操作
6	检查当前应用中的位置准确度。
7	如果对准确度仍不满意，那么就对整台机器人进行一次Absolute Accuracy校准。具体请参见CalibWare的文件。

机械臂的更换

若不更换控制柜就更换机器人的机械臂，那么就需更新该控制柜中的Absolute Accuracy参数，并将机器人重新对准围笼。正如[第114页的更改校准数据](#)所述，将新换机器人的校准参数载入相应的控制器后，便可更新Absolute Accuracy参数。确保既载入了相应的校准数据，又激活了Absolute Accuracy。

至于如何将新换机器人对准围笼，则取决于安装时选择的机器人对准技术。如果新换机器人的安装销对准了围笼，那么无需作进一步对准，仅需将该机器人放在这些销上即可。如果用某种机器人程序来对准新换机器人，那么就需测量围笼的固定装置以及在数个位置上对该机器人进行测量（为达到最好效果，请使用原机器人所用的程序）。具体请参见[第130页的测量机器人的对准情况](#)。

3 Motion performance

3.1.5.2 丧失准确度

3.1.5.2 丧失准确度

原因和行动

机器人发生碰撞或遭遇明显的温度波动后往往会丧失准确度。

需确定相关错误的原因，并采取适当的行动。

如果	...请...
未恰当校准此工具。	如果TCP已被改变，则重新进行校准。
未正确定义此工具负载	执行Load Identification，以确保所激活工具的质量、重心和惯量无误。
旋转变压器偏移不再有效	<ol style="list-style-type: none">1 通过检查轴刻度来查看机器人是否正确地立在起始位置上。2 如果指示器未对准，则将机器人移到正确的位置上，然后更新转数计数器。3 如果指示器已近乎对准但存在错误，则用校准单摆重新校准。
机器人与固定装置间的关系已发生变化	<ol style="list-style-type: none">1 将机器人移到相应固定装置的某个预定义位置上，以便开展检查。2 目测评估偏差是否过大。3 如果过大，则将机器人重新对准固定装置。
该机器人的结构已发生变化	<ol style="list-style-type: none">1 目测评估该机器人是否受损。2 如果受损，则更换整支机械臂，或更换受影响的臂，又或重新校准受影响的臂。

3.1.6 补偿理论

3.1.6.1 错误来源

错误类型

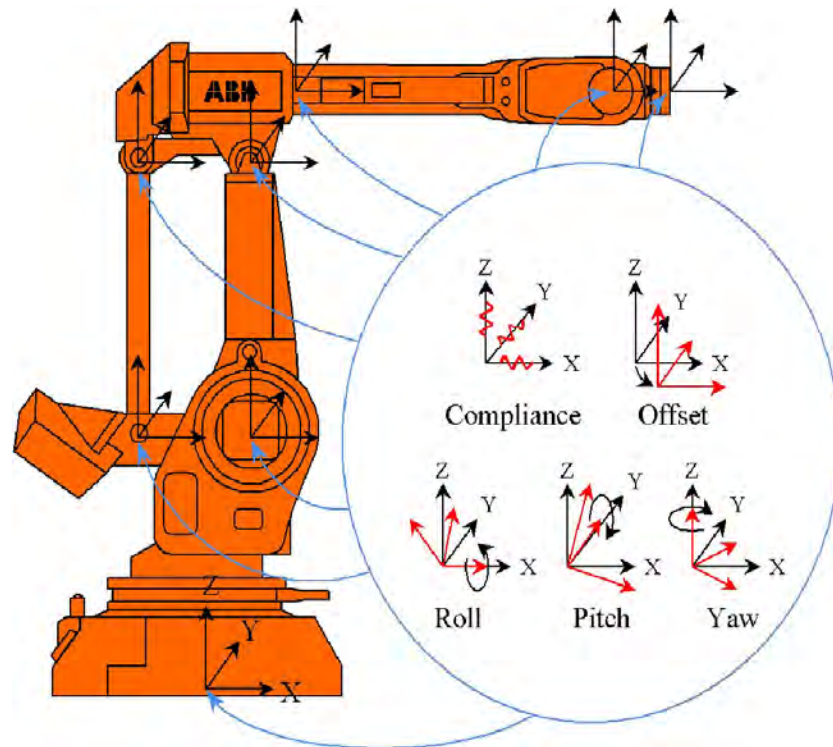
控制器中的已补偿错误源自机器人组成部分的机械容限。下文的插图详细展示了它们的一个子集。

机器人的自重以及当前有效负载导致了各种合规性错误。这些错误取决于相关负载的重力和特点。使用Load Identification可最高效地补偿这些错误（具体请参见操作员手册 - 带 *FlexPendant* 的 *IRC5*）。

机器人轴的位置或方位偏差导致了各种运动学错误。此类错误与负载无关。

图示

每个关节都可能发生若干种错误。



en0300000232

3 Motion performance

3.1.6.2 Absolute Accuracy补偿

3.1.6.2 Absolute Accuracy补偿

简介

用“虚假目标点”来补偿合规性错误和运动学错误。若了解机器人的偏移（即与预定位置之间的偏移程度），则可通过“下令该机器人前往某个虚假目标点”的方式来补偿 *Absolute Accuracy*。

补偿会作用在笛卡儿坐标上的某个机器人目标点上，而不是作用在单个关节上。这意味着该点就是TCP（标有下图中的箭头）在获得正确补偿后的位置。

所需位置

下图展示了您想设置的机器人位置。



xx0300000225

偏移所致位置

下图展示了机器人在没有 *Absolute Accuracy* 的情况下能实现的位置。机器人臂和负载的重量会使机器人发生偏移。注意图中的偏移有所夸大。



xx0300000227

虚假目标点

为了获得所需位置，*Absolute Accuracy* 会计算一个虚假目标点。当您输入一个所需位置时，系统会将其重新计算成一个虚假目标点，而该目标点经偏移后便能处于所需位置。



xx0300000226

下一页继续

已补偿的位置

实际位置将与您所需的位置如出一辙。从用户角度来说，您既不会注意到虚假目标点，也不会注意到偏移。机器人的一举一动将如同完全没有偏移一样。



xx0300000224

3 Motion performance

3.1.7.1 ABB校准进程

3.1.7 Absolute Accuracy机器人的准备

3.1.7.1 ABB校准进程

概述

本节描述了交付机器人前ABB公司在每台Absolute Accuracy机器人（不论是哪种类型或哪一系列的机器人）上执行的校准进程。

此进程可分为四步：

- 1 旋转变压器偏移校准
- 2 Absolute Accuracy校准
- 3 保存在系列测量板上的校准数据
- 4 Absolute Accuracy验证
- 5 生成出厂证书

旋转变压器偏移校准

通过此旋转变压器偏移校准进程来校准相应的旋转变压器偏移参数。

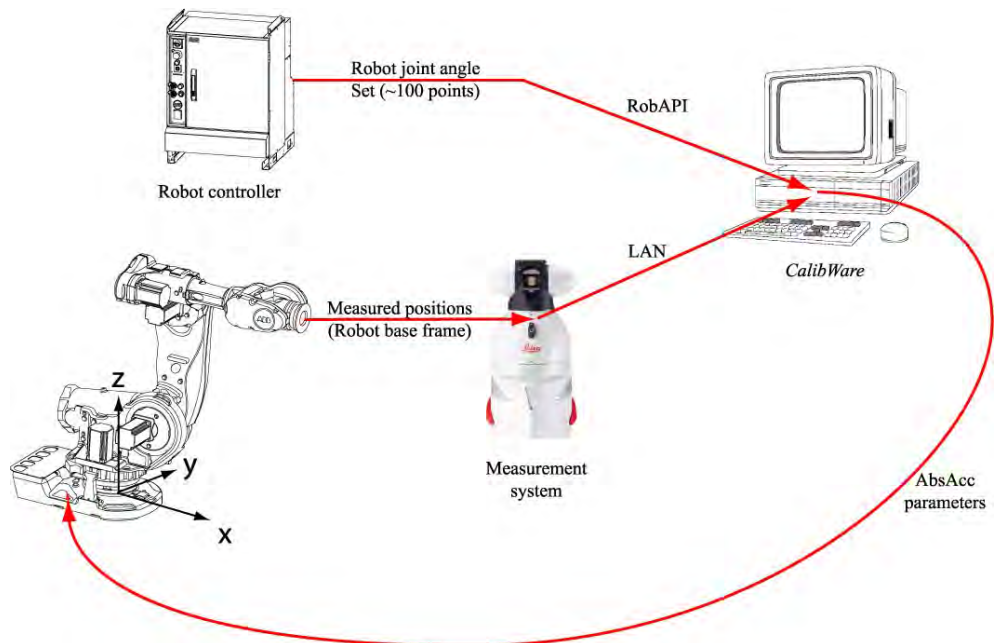
各台机器人的产品手册描述了具体做法。

Absolute Accuracy校准

由于方法的可重复性对两种进程都具有重要意义，因此在旋转变压器校准时要首先进行Absolute Accuracy校准。

按最大负载来校准每台机器人，从而确保检测到正确的补偿参数（若按低负载校准，则可能无法恰当确定机器人的灵活性参数）该进程会让机器人运行100种联合目标点姿态，并测量每个对应测量点的坐标。请向CalibWare校准核心输入各种姿态和测量的清单，然后创建一套机器人补偿参数。

CalibWare的文件描述了这方面的具体做法。



en0300000248

下一页继续

Absolute Accuracy验证

将这些参数载入控制器，然后将它们激活。机器人随后会运行一套共50种的机器人目标点姿态。系统会测量每种姿态，然后标定相应偏差。

CalibWare的文件描述了这方面的具体做法。

不同类型的机器人有不同的验收要求，但通常来说，90%的姿态（皆非单一姿态）都必须具备1毫米以下的绝对偏差。

补偿参数与出厂证书

这些补偿参数会保存在该机器人的系列测量板上（具体请参见[第125页的配置参数](#)）。

创建一份出厂证书，其代表了校准与验证序列所用的Absolute Accuracy测量协议（具体请参见[第124页的出厂证书](#)）。

3.1.7.2 出厂证书

关于出厂证书

该出厂证明由以下关键信息组成：

- 机器人信息（机器人类型，序列号）
- 准确度信息（精确点误差分布的最大偏差、平均偏差和标准偏差）
- 工具信息（TCP、指令和重心）
- 对测量协议的描述（测量与校准系统、点数和测量点的位置）

出厂证书示例

ABSOLUTE ACCURACY BIRTH CERTIFICATE																													
Calibration Date	January 7, 2002	Robot Version	IRB6400R_25_150																										
		Robot Serial Number	64-15677																										
Calibration and Verification Information																													
<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>Accuracy Information</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Measured verification points</td> <td style="text-align: center;">50</td> </tr> <tr> <td>Average Absolute Error (mm)</td> <td style="text-align: center;">0.47</td> </tr> <tr> <td>Maximum Absolute Error (mm)</td> <td style="text-align: center;">1.27</td> </tr> <tr> <td>Standard Deviation (mm)</td> <td style="text-align: center;">0.23</td> </tr> <tr> <td>Within Specification (≤ 1 mm) (%)</td> <td style="text-align: center;">98%</td> </tr> </table> <p>Tool Information</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th></th> <th style="text-align: center;">X</th> <th style="text-align: center;">Y</th> <th style="text-align: center;">Z</th> </tr> </thead> <tbody> <tr> <td>TCP (mm)</td> <td style="text-align: center;">185.7979736</td> <td style="text-align: center;">-272.594</td> <td style="text-align: center;">545.146</td> </tr> <tr> <td>COG (mm)</td> <td style="text-align: center;">106.4999998</td> <td style="text-align: center;">29.1</td> <td style="text-align: center;">351.9</td> </tr> <tr> <td>Mass (kg)</td> <td style="text-align: center;">144.6000061</td> <td></td> <td></td> </tr> </tbody> </table> </div> <div style="width: 50%;"> <p>GENERAL ABSOLUTE ACCURACY INFORMATION</p> <ol style="list-style-type: none"> 1) Default values Default number of Calibration Points = 100 Default number of Verification Points = 50 2) Measurement system Leica LTD500. Maximum error = 10µm/m. 4) Calibration software ABB CalibWare (PC), ABB S4Ident (S4cplus) 5) Calibration process Calibration performed via measurement of the default number of calibration points by the stated measurement system. The stated calibration software is used to generate robot error parameters, robot base to measurement system relationship, and TCP coincidental with Measurement Point (MP) location. 6) TCP/MP information TCP pre-defined by CMM measurement. Tool interface parameters determined for tool to robot flange mounting. Customer TCP should be calibrated before production start. A CMM or similar measured TCP can be used in conjunction with a tool interface calibration. 7) Base Frame Alignment Measurement system pre-aligned to physical mounting points. </div> </div>				Measured verification points	50	Average Absolute Error (mm)	0.47	Maximum Absolute Error (mm)	1.27	Standard Deviation (mm)	0.23	Within Specification (≤ 1 mm) (%)	98%		X	Y	Z	TCP (mm)	185.7979736	-272.594	545.146	COG (mm)	106.4999998	29.1	351.9	Mass (kg)	144.6000061		
Measured verification points	50																												
Average Absolute Error (mm)	0.47																												
Maximum Absolute Error (mm)	1.27																												
Standard Deviation (mm)	0.23																												
Within Specification (≤ 1 mm) (%)	98%																												
	X	Y	Z																										
TCP (mm)	185.7979736	-272.594	545.146																										
COG (mm)	106.4999998	29.1	351.9																										
Mass (kg)	144.6000061																												
<div style="display: flex; justify-content: space-between; margin-bottom: 10px;"> <div style="width: 45%;"> <p>Signature of acceptance (ABB SEROP)</p> <div style="border: 1px solid black; height: 40px; margin-top: 5px;"></div> </div> <div style="width: 50%;"> <p>Signature of acceptance (ABB FAC)</p> <div style="border: 1px solid black; height: 40px; margin-top: 5px;"></div> </div> </div>																													



www.abb.com/robots

xx0300000230

3.1.7.3 配置参数

关于补偿参数

所有Absolute Accuracy机器人都随附了一套补偿参数。由于Absolute Accuracy校准中的旋转变压器偏移校准浑然一体，因此旋转变压器偏移参数也保存在该机器人的系列测量板上。

补偿参数

The compensation parameters包含了以下各段：

- ROBOT_CALIB
- ARM_CALIB
- JOINT_CALIB
- PARALLEL_ARM_CALIB
- TOOL_INTERFACE
- MOTOR_CALIB

ROBOT_CALIB段定义了校准结构的最高等级。系统的默认设置是“uncalib”，这会导致Absolute Accuracy停用。“r1_calib”实例会通过指定“-absacc”旗标的方式激活Absolute Accuracy功能。此外还选择了一个工具接口（此处为“r1_tool”。注意必须手动激活Absolute Accuracy，具体请参见第114页的激活Absolute Accuracy。

系统保存了段ARM_CALIB、JOINT_CALIB、PARALLEL_ARM_CALIB和MOTOR_CALIB，并会在激活Absolute Accuracy功能时自动选择这些段。您可输入新的配置文件来更改相应的参数值，但关键词必须原样保留。改动这些关键词会导致配置文件损坏。

若想查看这些补偿参数，则可创建一份备份文件，然后读取moc.cfg文件。

补偿参数示例（和备份moc.cfg中的内容一模一样）

```
MOC:CFG_1.0::
# ROBOT_CALIB - ?
ROBOT_CALIB:
-name "r1_calib"
-use_tool_interface "r1_tool" -absacc

# ARM_CALIB - ?
ARM_CALIB:
-name "robl_1"
-error_offset_x 0.0000000 -error_offset_y 0.0000000 -error_offset_z
0.0000000 \
-error_roll 0.0000000 -error_pitch 0.0000000 -error_jaw 0.0000000
-arm_compliance_y 0.0000000

-name "robl_2" \
-error_offset_x 0.0002967 -error_offset_y 0.0000000 -error_offset_z
0.0000000 \
-error_roll 0.0001903 -error_pitch -0.0003469 -error_jaw 0.0000000

-name "robl_3" \
```

下一页继续

3 Motion performance

3.1.7.3 配置参数

续前页

```
-error_offset_x 0.0000000 -error_offset_y 0.0000000 -error_offset_z
0.0005485 \
-error_roll 0.0000537 -error_pitch 0.0006959 -error_jaw 0.0003361
-arm_compliance_x 0.0000000 -arm_compliance_z 0.0000000

-name "robl_4" \
-error_offset_x 0.0000000 -error_offset_y -0.0003586 -error_offset_z
0.0004580 \
-error_roll 0.0000965 -error_pitch 0.0000000 -error_jaw -0.0002578

-name "robl_5" \
-error_offset_x -0.0005467 -error_offset_y 0.0000000 -error_offset_z
0.0000032 \
-error_roll 0.0000000 -error_pitch 0.0009360 -error_jaw -0.0002367

-name "robl_6" \
-error_offset_x 0.0000000 -error_offset_y -0.0000449 -error_offset_z
-0.0000365 \
-error_roll 0.0000000 -error_pitch 0.0000000 -error_jaw -0.0002168

# JOINT_CALIB - ?
JOINT_CALIB:
-name "robl_1" -compl 0.00000000
-name "robl_2" -compl 0.00000004
-name "robl_3" -compl 0.00000107
-name "robl_4" -compl 0.00000257
-name "robl_5" -compl 0.00000490
-name "robl_6" -compl 0.00000941

# PARALLEL_ARM_CALIB - ?
PARALLEL_ARM_CALIB:
-name "robl_2" -error_length 0.0004324
-name "robl_3" -error_length -0.0000744

# TOOL_INTERFACE - ?
TOOL_INTERFACE:
-name "rl_tool" -compl 0.0 -mass 0.0 -mass_centre_x 0.0 \
-offset_x -0.0000465 -offset_y 0.0011064 -offset_z -0.0005255 \
-orient_u0 1.0 -orient_u1 0.0 -orient_u2 0.0 -orient_u3 0.0

# MOTOR_CALIB - ?
MOTOR_CALIB:
-name "robl_1" -valid_com_offset -cal_offset 1.301100
-valid_cal_offset
-name "robl_2" -valid_com_offset -cal_offset 3.422110
-valid_cal_offset
-name "robl_3" -valid_com_offset -cal_offset 5.057730
-valid_cal_offset
-name "robl_4" -valid_com_offset -cal_offset 3.584140
-valid_cal_offset
-name "robl_5" -valid_com_offset -cal_offset 3.556740
-valid_cal_offset
```

下一页继续

```
-name "robl_6" -valid_com_offset -cal_offset 4.180770  
-valid_cal_offset
```

3 Motion performance

3.1.8.1 概述

3.1.8 围笼的对准

3.1.8.1 概述

关于围笼的对准

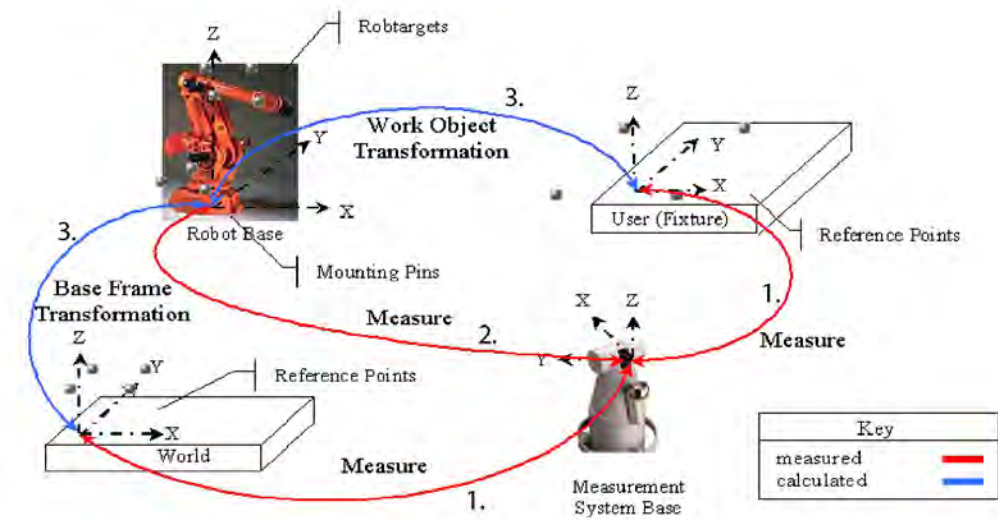
系统会确定Absolute Accuracy机器人的相关补偿参数（从物理基板到机器人工具）。对许多应用而言，这已足以像其它机器人那样来使用该机器人。不过Absolute Accuracy机器人往往会对准其围笼中的坐标，而本节就描述了这种无返回值对准程序。更详细的说明则请参见CalibWare的文件。

无返回值对准程序

要想让机器人在整个机器人围笼内都保持准确，就得正确安装该机器人。简而言之，这涉及到：

	操作	描述
1	测量固定装置的对准情况	确定相关测量系统与相关固定装置之间的关系。具体请参见第129页的测量固定装置的对准情况。
2	测量机器人的对准情况	确定相关测量系统与相关机器人之间的关系。具体请参见第130页的测量机器人的对准情况。
3	计算框架关系	确定相应的关系（比如相关机器人与相关固定装置之间的关系。具体请参见第131页的框架关系。
4	校准工具	确定相关机器人工具与其它围笼部件之间的关系。具体请参见第132页的工具校准。

图示



en0300000239

3.1.8.2 测量固定装置的对准情况

关于固定装置的对准情况

将一个固定装置定义为与特定坐标系有关的一种围笼部件。为了确保Absolute Accuracy，需要有一种准确的关系来实现机器人与固定装置之间的相互作用。

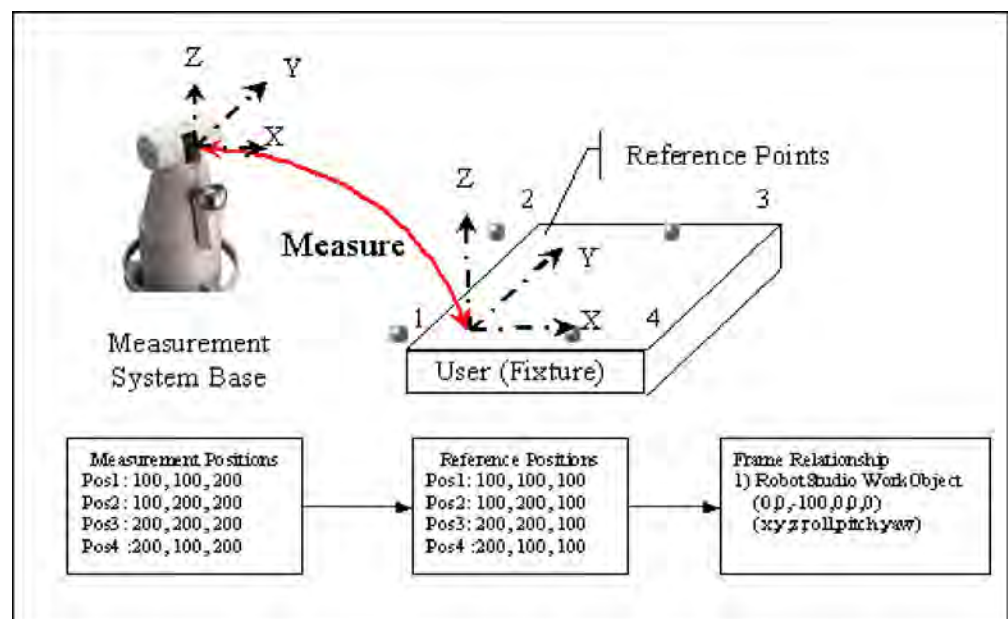
Absolute Accuracy固定装置必须至少有三个（四个更好）参考点，且明确标注每个点的位置信息。

固定装置的无返回值测量程序

按下列步骤来对准固定装置：

- 1 在对准软件（如CalibWare等）中输入参考点的名称和位置。
- 2 测量相关参考点，并赋予相同的名称。
- 3 用对准软件来匹配相关引用项和所测点，然后确定关系框架。所有测量系统都支持这种转变格式。

图示



en0300000237

3 Motion performance

3.1.8.3 测量机器人的对准情况

3.1.8.3 测量机器人的对准情况

选择方法

可通过两种途径来确定相关测量系统与相关机器人之间的关系：

无返回值对准程序	描述
对准物理基座	相当于按“根据特定机器人用户手册中详述的参考位置来测量和对准物理基座的销”这一方式来对准此固定装置。
对准理论基座	测量若干种机器人姿态，并用对准软件来确定该机器人的对准情况。

对准物理基座

像固定装置那样对准机器人的优点在于简单易行——将相关机器人视为围笼中的另一个固定装置，然后相应测量其基点即可。而这种做法的缺点在于：之后将机器人放到各销上时若存在小错，则可能因机器人的伸展范围而形成较大的TCP错误（即未校准机器人的放置情况）。

若要确定参考点的坐标，则需查阅此类机器人的产品手册。

一旦测得正确的点，系统就会用对准软件来确定相关测量系统与机器人基座之间的框架关系。

对准理论基座

让机器人对准理论基座的优点在于：可消除安装机器人时产生的一切错误，此外对准进程中还会详述所测点的机器人准确度，从而确认正确的Absolute Accuracy功能。这种做法的缺点在于：必须创建一则机器人程序（用CalibWare手动或自动创建），并对机器人进行测量（理想情况是使用正确工具进行测量，但也可把校准TCP作为这则无返回值程序的一部分）。

一旦测得正确的点，系统就会用对准软件来确定相关测量系统与机器人基座之间的框架关系。

3.1.8.4 框架关系

关于框架关系

一旦测得相关测量系统与其它所有围笼部件之间的关系，就能确定各围笼部件之间的关系。

全局坐标系和相关机器人之间的关系应保存在机器人基座中。相关机器人与相关固定装置之间的关系则应保存在`workobject`数据类型中。

由于以相对于测量系统的方式测量了全局和机器人，因此该测量系统最初是处于激活状态的相应坐标系。

确定机器人基座

用一套标准测量系统软件来确定全局坐标中的机器人基座：

- 1 将全局坐标系设置成激活状态（原点）。
- 2 读取机器人基本框架的坐标（此时是相对于全局的坐标）。

将该机器人设置成激活状态，然后读取固定装置框架的坐标，从而以类似方式确定固定装置的关系。

3.1.8.5 工具校准

关于工具校准

Absolute Accuracy机器人补偿参数会被计算为独立工具，这将使所有正确预定义了TCP的工具都与机器人法兰连接起来，且无需重新校准它们就能使用这些工具。不过由于存在未考虑到工具与机器人间的连接或工具灵活性等问题，因此实际上难以用Coordinate Measurement Machine (CMM)等来进行正确的TCP校准。

为了确保机器人具有最佳准确度，用户最好定期校准每件工具。

无返回值工具校准程序

下文详述了建议采用的无返回值工具重新校准程序：

- SBCU (Single Beam Calibration Unit) 比如针对电弧焊应用或点焊应用的ABBBullsEye。
- 4、5或6点（控制器上可用的工具中心点校准例程）等几何校准。可用一套测量系统来确保所用的单个点是准确的。
- RAPID工具校准例程：MToolTCPCalib（校准移动工具的TCP）、SToolTCPCalib（校准固定工具的TCP）、MToolRotCalib（校准移动工具的旋度）以及SToolRotCalib（校准固定工具的TCP和旋度）。
- 使用理论数据，比如来自CAD模型的理论数据。



提示

由于Absolute Accuracy型号使用了相应的工具负载特点，因此至关重要的一点就是所有参数都要尽量准确。可用Load Identification来高效地确定各项工具负载特点。

3.2 Advanced robot motion [687-1]

关于Advanced robot motion

您可通过选项*Advanced robot motion*来访问：

- *Advanced Shape Tuning*方面的信息请参见第134页的*Advanced Shape Tuning*【包括在687-1中】。
- 用RAPID更改*Motion Process Mode*，具体请参见第141页的*Motion Process Mode*【包括在687-1中】。
- *Wrist Move*方面的信息请参见第148页的*Wrist Move*【包括在687-1中】。

3 Motion performance

3.3.1 关于Advanced Shape Tuning

3.3 Advanced Shape Tuning 【包括在687-1中】

3.3.1 关于Advanced Shape Tuning

目的

*Advanced Shape Tuning*的作用是减少机器人关节摩擦产生的路径偏差。

*Advanced Shape Tuning*有助于小圆环等的低速切割（10到100毫米 / 秒）——这种情况下机器人的关节摩擦通常会带来0.5毫米的路径偏差。通过微调控制器中摩擦模型的参数，系统可将这种路径偏差减少到该机器人的可重复性水平上（比如中型机器人为0.1毫米）。

其中包括

RobotWare选项和*Advanced robot motion*中都包括了*Advanced Shape Tuning*，您可由此访问：

- 指令FricIdInit、FricIdEvaluate和FricIdSetFricLevels，这些指令会自动优化已编程路径所需的关节摩擦模型参数。
- 用于手动微调关节摩擦参数的系统参数*Friction FFW On*、*Friction FFW level*和*Friction FFW Ramp*。
- 微调类型tune_fric_lev和tune_fric_ramp，可搭配指令TuneServo使用。

基本方法

此处简述了最常见的Advanced Shape Tuning用法：

- 1 将系统参数*Friction FFW On*设置成TRUE。具体请参见[第138页的系统参数](#)。
- 2 用指令FricIdInit和FricIdEvaluate进行关节摩擦等级的自动微调。具体请参见[第135页的自动微调摩擦](#)。
- 3 用指令FricIdSetFricLevels来补偿相关摩擦。

3.3.2 自动微调摩擦

关于自动微调摩擦

用指令`FricIdInit`和`FricIdEvaluate`可自动微调一台机器人的关节摩擦等级。这些指令会微调每个关节在特定移动序列中的摩擦等级。

通过指令`FricIdSetFricLevels`把自动微调等级应用到摩擦补偿上。

程序执行

若要对一段移动序列做自动微调，该序列就必须从指令`FricIdInit`开始，到指令`FricIdEvaluate`结束。当程序执行到`FricIdEvaluate`时，机器人将重复该移动序列，直至找到每个关节轴的最佳摩擦等级为止。每次反复都包括一次后退运动和一次前进运动（均沿所编程的路径运动）。为了得出正确的关节摩擦等级，该序列通常必须重复20到30次才行。

当程序指针处在指令`FricIdEvaluate`上时，若相关程序因故停止执行并随之重启，那么所得结果就无效。这样一来，停止后就必须从开头处重启摩擦识别。

一旦找到正确的摩擦等级，就必须用指令`FricIdSetFricLevels`来设置它们，否则系统就不会使用它们。注意是针对`FricIdInit`与`FricIdEvaluate`之间的特定移动来微调相应的摩擦等级。至于机器人工作区域中的其它地方，则需另做一次微调来获取正确的摩擦等级。

这些指令方面的详细说明请参见技术参考手册 - *RAPID*指令、函数和数据类型。

限制

摩擦微调存在以下限制：

- 摩擦微调无法与同步移动组合起来，即是不允许在`FricIdInit`与`FricIdEvaluate`之间实现`SyncMoveOn`。
- 针对所作摩擦微调的移动序列必须从一个精确点开始，到一个精确点结束。若非如此，系统则会在微调期间自动插入精确点。
- 自动微调摩擦仅对TCP机器人有用。
- 一次只能对一台机器人进行自动微调关节摩擦。
- 最多可微调至500%。如果这还不够，那么就为参数`Friction FFW Level`设置更大的值。具体请参见第139页的[用估计值启动](#)。
- 自动微调摩擦时将无法用Test Signal Viewer看到任何测试信号。
- `FricIdInit`与`FricIdEvaluate`之间的移动序列不能超过10秒。



注意

若要使用Advanced Shape Tuning，就必须将参数`Friction FFW On`设置成TRUE。

示例

此例展示了如何编写一条封装了摩擦微调的切割指令。系统会在首次执行该指令时（没有计算出的摩擦参数）微调摩擦。在微调期间，机器人将沿着所编程的路径来回移动。这种动作大约需要重复25次。

下一页继续

3 Motion performance

3.3.2 自动微调摩擦

续前页

在所有后续执行过程中，相关的摩擦等级都会被设置成第一次执行时确认的微调值。
可用指令CutHole来单独微调每个孔的摩擦。

```
PERS num friction_levels1{6} := [9E9,9E9,9E9,9E9,9E9,9E9];
PERS num friction_levels2{6} := [9E9,9E9,9E9,9E9,9E9,9E9];

CutHole p1,20,v50,tool1,friction_levels1;
CutHole p2,15,v50,tool1,friction_levels2;

PROC CutHole(robtarget Center, num Radius, speeddata Speed, PERS
  tooldata Tool, PERS num FricLevels{*})
  VAR bool DoTuning := FALSE;

  IF (FricLevels{1} >= 9E9) THEN
    ! Variable is uninitialized, do tuning
    DoTuning := TRUE;
    FricIdInit;
  ELSE
    FricIdSetFricLevels FricLevels;
  ENDIF

  ! Execute the move sequence
  MoveC p10, p20, Speed, z0, Tool;
  MoveC p30, p40, Speed, z0, Tool;

  IF DoTuning THEN
    FricIdEvaluate FricLevels;
  ENDIF
ENDPROC
```



注意

真实的程序中将包括“在微调阶段前停用切割设备”。

3.3.3 手动微调摩擦

概述

可以手动微调一台机器人的关节摩擦（而不是自动微调摩擦）。可用指令 `TuneServo` 来微调每个关节的摩擦等级。本节描述了其具体做法。

通常无需更改摩擦增减率。



注意

若要使用 Advanced Shape Tuning，就必须将参数 *Friction FFW On* 设置成 TRUE。

微调类型

将一种微调类型作为指令 `TuneServo` 的一个自变数。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的微调类型。

Advanced Shape Tuning 显然使用了两种微调类型：

微调类型	描述
TUNE_FRIC_LEV	在用自变数 TUNE_FRIC_LEV 调用指令 <code>TuneServo</code> 后，便可在执行程序时调节一个机器人关节的摩擦等级。为参数 <i>Friction FFW Level</i> 所定义的摩擦等级给出一个百分数（1到500之间）。
TUNE_FRIC_RAMP	在用自变数 TUNE_FRIC_RAMP 调用指令 <code>TuneServo</code> 后，便可在执行程序时调节完全摩擦补偿下所能达到的电机轴速。为参数 <i>Friction FFW Ramp</i> 所定义的摩擦等级给出一个百分数（1到500之间）。通常无需微调摩擦坡。

配置摩擦等级

设置每个机器人关节的摩擦等级。一次对一个关节采取以下步骤：

	操作
1	通过让机器人运行其任务最急需的部分（最高级的形状）来测试该机器人。如果应使用机器人进行切割，则在测试中使用与生产时相同的工具来进行切割。观测路径偏差，并测试是否需要增加或减少各个关节摩擦等级。
2	用 <i>RAPID</i> 指令 <code>TuneServo</code> 和微调类型 TUNE_FRIC_LEV 来微调相应的摩擦等级。 <i>Friction FFW Level</i> 值提供了相关等级的百分数。 示例：将摩擦等级增大20%的指令如下： <code>TuneServo MHA160R1, 1, 120 \Type:= TUNE_FRIC_LEV;</code>
3	重复步骤1和2，直到您对路径偏差满意为止。
4	可将最终微调值传输给相应的系统参数。 示例： <i>Friction FFW Level</i> 为0.5，最终微调值（TUNE_FRIC_LEV）为120%。将 <i>Friction FFW Level</i> 设置成0.6，将微调值设置成100%（默认值）——两者是等同的。



提示

最多可微调至500%。如果这还不够，那么就为参数 *Friction FFW Level* 设置更大的值。具体请参见第139页的设置微调系统参数。

3 Motion performance

3.3.4.1 系统参数

3.3.4 系统参数

3.3.4.1 系统参数

关于系统参数

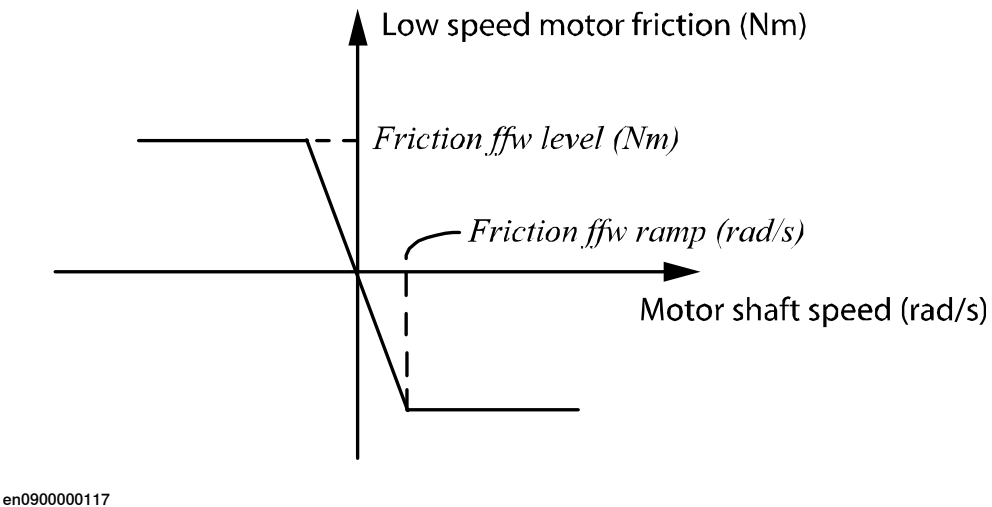
此处简述了Advanced Shape Tuning中的每个参数。更多信息请参见技术参考手册 - 系统参数中的各个参数。

Friction Compensation / Control Parameters

除机器人IRB 1400与IRB 1410（它们属于主题*Motion*下的类型*Control Parameters*）外，这些参数均属于主题*Motion*下的类型*Friction Compensation*。

参数	描述
Friction FFW On	当 <i>Friction FFW On</i> 被设置成TRUE时，Advanced Shape Tuning便会处于激活状态。
Friction FFW Level	<i>Friction FFW Level</i> 是相关机器人关节的摩擦等级。参见下图。
Friction FFW Ramp	<i>Friction FFW Ramp</i> 为摩擦力达到 <i>Friction FFW Level</i> 定义的摩擦等级时的机器人电机轴速度。参见下图。 通常无需更改 <i>Friction FFW Ramp</i> 。

图示



3.3.4.2 设置微调系统参数

自动微调时很少需要更改系统参数

对自动微调而言，如果将相应的摩擦等级保存在永久数组中，则可在上电失败后保留微调项。此外自动微调还能为不同的机器人移动序列设置不同的微调等级（系统参数则做不到这一点）。在使用自动微调时，除非默认值几乎都被“关闭”，否则无需更改相关系统参数。具体请参见第139页的用估计值启动。

将微调项传输给系统参数

使用手动微调时，相关微调值会被重置为上电失败时的默认项（100%）。不过系统参数设定是永久性的。

如果采用临时微调，则仅在执行一部分程序时才会有效，且不宜向外传输。

若要将摩擦等级微调值（TUNE_FRIC_LEV）传输给参数*Friction FFW Level*，则请遵循下列步骤：

	操作
1	在RobotStudio中打开配置编辑器（Motion主题），然后选择类型Friction comp（机器人IRB 1400与IRB 1410除外——它们属于类型Control parameters）。
2	用 <i>Friction FFW Level</i> 乘以相应的微调值。将该值设置成新的 <i>Friction FFW Level</i> ，然后将该微调值（TUNE_FRIC_LEV）设置成100%。 示例： <i>Friction FFW Level</i> 为0.5，最终微调值（TUNE_FRIC_LEV）为120%。将 <i>Friction FFW Level</i> 设置成0.6（ 1.20×0.5 ），将微调值设置成100%（默认值）——两者是等同的。
3	重启控制器，从而使所做改动生效。

用估计值启动

参数*Friction FFW Level*将作为微调启动值。如果该值与正确值相距甚远，那么可能就无法将其微调成正确值。不过*Friction FFW Level*大多数情况下都会被默认设置成一个基本正确的数值，因此不太可能发生这种情况。

如果*Friction FFW Level*出于某种原因而与正确值相距甚远，则可将其改为一个新估计值。

	操作
1	在RobotStudio中打开配置编辑器（Motion主题），然后选择类型Friction comp（机器人IRB 1400与IRB 1410除外——它们属于类型Control parameters）。
2	将参数 <i>Friction FFW Level</i> 设置成一个估计值。请勿将该数值设置成0（零），不然将无法进行微调。
3	重启控制器，从而使所做改动生效。

3 Motion performance

3.3.5 RAPID组件

3.3.5 RAPID组件

关于RAPID部件

此处概述了 *Advanced Shape Tuning* 中的所有指令、函数和数据类型。
有关更多信息，请参阅 技术参考手册 - *RAPID* 指令、函数和数据类型。

指令：

指令：	描述
FricIdInit	发起摩擦识别
FricIdEvaluate	评价摩擦识别
FricIdSetFricLevels	在摩擦识别后设置摩擦等级

函数

Advanced Shape Tuning 中不包括任何函数。

数据类型

Advanced Shape Tuning 中不包括任何数据类型。

3.4 Motion Process Mode 【包括在687-1中】

3.4.1 关于Motion Process Mode

目的

Motion Process Mode的作用是简化专用微调，也就是优化特定应用中的机器人性能。对大多数应用而言，默认模式就是最佳模式。

可用的运动进程模式

运动进程模式由一套特定的机器人微调参数组成。每套微调参数（也就是每种模式）都会针对特定的应用等级来优化机器人的微调。

预定义了下列模式：

- *Optimal cycle time mode* – 此模式产生可能的最短循环时间，通常是默认模式。
- *Accuracy mode* – 此模式提高了路径准确度。相较*Optimal cycle time mode*，循环时间将稍稍增加。这是提高小型和中型机器人（比如，IRB 2400和IRB 2600）路径准确度的建议选项。
- *Low speed accuracy mode* – 此模式提高了路径准确度。相较*Accuracy mode*，循环时间将稍稍增加。这是提高大型机器人（比如，IRB 4600）路径准确度的建议选项。
- *Low speed stiff mode* – 建议在最大伺服器刚性具有重要意义的接触应用中使用该模式。此外也可用于某些想尽量减少路径波动的低速应用。该模式的周期时间要久于*Low speed accuracy mode*。

也有四种模式可供应用程序特定用户调整使用：

- *MPM User mode 1 – 4*

模式选择

系统会自动选择相应的默认模式，不过用户可在类型*Robot*的系统参数*Use Motion Process Mode*中更改此模式。

只有当安装了选项*Advanced Robot Motion*时，才能用RAPID来更改*Motion Process Mode*。只有当机器人直立不动时才能更改该模式，否则就会强制使用一个精确点。

下例展示了RAPID指令*MotionProcessModeSet*的典型用法。

```
MotionProcessModeSet OPTIMAL_CYCLE_TIME_MODE;
! Do cycle-time critical movement
MoveL *, vmax, ...;
...

MotionProcessModeSet ACCURACY_MODE;
! Do cutting with high accuracy
MoveL *, v50, ...;
...
```

限制

- *Motion Process Mode*方案目前可用于所有六轴和七轴机器人，但上漆机器人除外。

下一页继续

3 Motion performance

3.4.1 关于Motion Process Mode

续前页

- *Mounting Stiffness Factor*仅能用于以下机器人：
IRB 120, IRB 140, IRB 1200, IRB 1520, IRB 1600, IRB 2600, IRB 4600, IRB 6620 (非LX) , IRB 6640, IRB 6700.
- 对于IRB 1410, 只有*Accset*和几何准确度参数可用。
- 下列机器人模型不支持使用*World Acc Factor* (即, 只允许*World Acc Factor* = -1) :
IRB 340, IRB 360, IRB 540, IRB 1400, IRB 1410

3.4.2 用户定义的模式

可用的微调参数

如果需要更具体的微调，部分调节参数可以在每个每个动作处理模式中可以修改。四个预定义模式和四个用户模式都可以修改。这样，用户就可以为特定应用进行专门校准。

下表对可用的微调参数作了简短说明。

- *Use Motion Process Mode Type* - 选择用户模式下的预定义参数。
- *Accset Acc Factor* - 更改加速度
- *Accset Ramp Factor* - 更改加速度增减率
- *Accset Fine Point Ramp Factor* - 更改精确点的减速度增减率
- *Joint Acc Factor* - 修改特定关节的加速。
- *World Acc Factor* - 如为正，则激活动态世界加速，典型值为1，如为-1则停止。
- *Geometric Accuracy Factor* - 如果减少，则增加geometric精确度。
- *Dh Factor* - 更改路径的平顺度（有效的系统带宽）
- *Df Factor* - 更改某一根轴的预测共振频率
- *Kp Factor* - 更改某一根轴的位置控制器的等效增益
- *Kv Factor* - 更改某一根轴的速度控制器的等效增益
- *Ti Factor* - 更改某一根轴的积分时间
- *Mounting Stiffness Factor X - I / O*描述了x方向上的机器人底座刚度
- *Mounting Stiffness Factor Y - I / O*描述了y方向上的机器人底座刚度
- *Mounting Stiffness Factor Z - I / O*描述了z方向上的机器人底座刚度

详细说明请参见技术参考手册 - 系统参数中的*Motion Process Mode*。

用RAPID微调各个参数

大部分参数也可以使用TuneServo和AccSet指令修改。



注意

所有参数设定都是在相对于预定义参数值进行调节。虽然运动进程模式和TuneServo/Accset指令可以一起使用，但我们还是建议要么选择运动进程模式，要么选择TuneServo/AccSet。

例 1

加速度的相对调节 = 【预定义AccSet Acc Factor】 * 【AccSet Acc Factor】 * 【AccSet指令加速度系数 / 100】

例 2

Kv的相对调节值 = 【预定义Kv Factor】 * 【Kv Factor】 * 【TuneServo(TYPE_KV)指令的微调值 / 100】

预定义参数值

如果机器人的类型不同，那么每种模式的预定义参数值也不相同。

对*Optimal cycle time mode*而言，通常所有预定义参数都会被设置成1.0。

下一页继续

3 Motion performance

3.4.2 用户定义的模式

续前页

对*Low speed accuracy mode*和*Low speed stiff mode*而言，系统会以降低*AccSet*和*Dh*参数的方式来提高移动的平顺度和路径的准确度，同时以更改*Kv Factor*、*Kp Factor*和*Ti Factor*的方式来提高伺服器的刚度。

某些机器人可能无法增大*Low speed accuracy mode*和*Low speed stiff mode*中的*Kv Factor*。在调节*Kv Factor*时，请始终小心行事和观察增大后的电机噪声等级，且采用的数值请勿超过达到相关应用要求时所需的数值。若*Kp Factor*太高或*Ti Factor*太低，则都会因机械共振而使振动加剧。

*Accuracy Mode*使用动态世界加速限制(*World Acc Factor*)且增加了地理准确性(*Geometric Accuracy Factor*)以提升路径准确性。

由于*Df Factor*和*Mounting Stiffness Factors*的最佳值取决于具体的安装情况（比如安装机器人的底座的刚度），因此预定义模式下的这些参数会始终被设置成1.0。可用*TuneMaster*来优化这些参数。用户可在*TuneMaster*应用中找到更多信息，此外还要注意*Mounting Stiffness Factor*的限制。



警告

若*Motion Process Mode*参数设定有误，则可能造成振荡移动或扭矩，从而对机器人造成损伤。

3.4.3 关于机器人微调的一般信息

尽量缩短周期时间

若要获得最好的周期时间，则宜采用运动进程模式 *Optimal cycle time mode*。该模式通常为默认模式，用户仅需定义工具负载、有效负载和臂负载（若有）即可。一旦编写好机器人路径，ABB QuickMove 运动技术就会自动计算该路径上的最佳加速度和最佳速度，从而得出周期时间最短的时间优化型路径，于是便无需对加速度进行微调。改善周期时间的唯一途径是更改相关路径的几何结构或处理工作空间的其它区域。若需进行此类优化，可通过 RobotStudio 中的模拟来开展此类优化。

增加路径准确度和减少振动

对大多数应用来说，*Optimal cycle time mode* 可令路径准确度和振动方面的行为达到令人满意的程度，而其中依靠的就是 ABB TrueMove 运动技术。不过某些应用的确需要通过修改机器人的微调来改进准确度。本文之前已用 RAPID 程序中的 TuneServo 和 AccSet 指令做了这种微调。

动作处理模式的概念将简化此应用程序的微调，且四个预定义模式在很多情况下应该都会很有用，无需进一步调整。

下文就如何解决准确度问题提出了一些通用建议（假设用户已测试过默认选择 *Optimal cycle time mode*，并注意到了各种准确度问题）：

- 1 验证是否恰当定义了工具负载、有效负载和臂负载。
- 2 将检查工具和工艺设备连接在机器人的臂上。确保所有东西均已固定牢靠，且相关工具具有足够的硬度。
- 3 检查安装机器人的底座，具体请参见 [第145页的补偿底座的灵活性](#)。

补偿底座的灵活性

如果相关底座未达到机器人产品手册中的刚度要求，那儿就宜对底座灵活性进行补偿。参见机器人产品手册中的节“底座要求，最小共振频率”。

要么通过轴1和轴2的 *Df Factor* 实现，要么通过 *Mounting Stiffness Factor* 实现（取决于机器人的类型）。具体请参见 [第147页的限制](#)。

TuneMaster 的作用是查找 *Df Factor* / *Mounting Stiffness Factor* 的最佳值。系统随后会为使用的 *Motion Process Modes* 定义所得的 *Df Factor* / *Mounting Stiffness Factor*。



注意

如果底座达不到相关要求，那么即使做了所述补偿，也仍不免在一定程度上削弱准确度。如果底座硬度很差，那么可能就无法用 *Df Factor* / *Mounting Stiffness Factor* 来解决相关问题。

这种情况下必须改善底座或采用以下解决方案之一，比如 *low Df Factor* 的 *Optimal cycle time mode*、*Accset Acc Factor* 或 *Accset Fine Point Ramp Factor*（具体请参见取决于相关应用）。

若准确度仍需改善

- 对于，例如切割，应该使用 *Advanced Shape Tuning and Accuracy mode* / *Low speed accuracy mode*。动作模式同时取决于机器人类型与具体的应用。总的来说，对小型和中型机器人推荐使用 *Accuracy mode* (IRB 2400/2600)，而 *Low speed accuracy mode* 则推荐对大机器人使用。

下一页继续

3 Motion performance

3.4.3 关于机器人微调的一般信息

续前页

- 如果路径准确性仍需提高，则可以用微调参数来调节准确性模式，以下为部分示例：
 - 调节*Accuracy mode*实现更好的准确性：
 - 1) 减少*World Acc Factor*，例如从1到0.5。
 - 2) 减少*Dh Factor*到0.5或更低。注意低值*Dh factor*可能会改变高速下的角区域。
 - 调节*Low speed accuracy mode*实现更好的准确性：
 - 1) 设置*World Acc Factor*为1，并设置*Geometric Accuracy Factor*为0.1。
 - 2) 减少*Dh Factor*到0.5或更低。
- 编程设定的速度有时必须降低，以实现可能范围内的最好准确性（如在切削应用中）。例如，半径1 mm的圆圈不应编程设定高于20 mm/s的速度。
- 对于接触类应用，例如铣削和预压，推荐使用*Low speed stiff mode*此模式也对某些低速应用中的大机器人适用（最大100 mm/s），其中对最小路径波动有要求（如小于0.1 mm）。注意此模式的伺服调节非常刚性，且有些情况下*Kv Factor*可能由于电机震动和噪音需要降低。
- 如果需要减少精确点的过界和振动，则请采用*Optimal cycle time mode*，同时减少*Accset Fine Point Ramp Factor*或*Dh Factor*的值，直至问题得到解决为止。
- 如果在开始或结束重定方位时发生了准确度问题，则用更大的*pzone_ori*和*pzone_eax*来定义一个新区。即使系统中没有外轴，其数值也最好始终相同。同时也增大*zone_ori*。编程时请始终力求能平顺地重定方位。
- 最后要说的是，结束准确度微调后若需减少周期时间，则在RAPID程序的不同段中使用不同的运动进程模式。

3.4.4 附加信息

与TuneServo和AccSet相比的Motion Process Mode

运动进程模式简化了专用微调，并使用户能通过系统参数——而不是RAPID程序——来定义相关微调项。

总的来说，用户宜把运动进程模式作为解决准确度问题的首选。不过用户仍然能用RAPID程序中的TuneServo和AccSet指令来进行专用微调。

TuneServo和AccSet通常都不是最好的选择，不过也有例外，比如如果减少RAPID程序中某段的加速度就能准确度问题，并使也对周期时间做了优化，那么它们就是明智之选。在这种情况下，由于需要一个精确点来更改运动进程模式，因此更好的做法或许是使用无需精确点就能更改的AccSet。

限制

- *Motion Process Mode*方案目前可用于所有六轴和七轴机器人，但上漆机器人除外。
- *Mounting Stiffness Factor*仅能用于以下机器人：
IRB 120, IRB 140, IRB 1200, IRB 1520, IRB 1600, IRB 2600, IRB 4600, IRB 6620（非LX），IRB 6640, IRB 6700.
- 对于IRB 1410，只有Accset和几何准确度参数可用。
- 下列机器人模型不支持使用*World Acc Factor*（即，只允许*World Acc Factor* = -1）：
IRB 340, IRB 360, IRB 540, IRB 1400, IRB 1410

相关信息

信息，关于	请参阅
<i>Motion Process Mode</i> 参数的配置。	技术参考手册 - 系统参数
RAPID指令： <ul style="list-style-type: none"> • AccSet - 减少加速度 • MotionProcessModeSet - 设置运动进程模式 • TuneServo - 微调伺服器 	技术参考手册 - RAPID指令、函数和数据类型

3 Motion performance

3.5.1 Wrist Move介绍

3.5 Wrist Move【包括在687-1中】

3.5.1 Wrist Move介绍

目的

*Wrist Move*的作用是改善切割小尺寸几何结构时的路径准确度。对小孔等几何形状而言，机器人主轴（1到3）产生的摩擦效应往往会削弱其形状的视觉外观。一条关键思路是不控制机器人的TCP，而是通过腕的移动来控制激光束（或水射流、布线芯轴等）与切割面之间的交叉点。用户只需两根腕轴即可控制该交叉点（而不是用上所有的机器人轴），于是便尽量减少了相关路径上的摩擦效应。由程序员决定使用哪一对腕轴。

使用Wrist Move

RobotWare选项*Advanced robot motion*中包括了*Wrist Move*。

*Wrist Move*和RAPID指令*CirPathMode*与各种移动指令一同用于圆弧作业，即*MoveC*、*TrigC*和*CapC*等。由指令*CirPathMode*配合旗标*Wrist45*、*Wrist46*或*Wrist56*之一来激活腕移动模式。在激活这种模式后，所有后续*MoveC*指令都将产生一次腕移动。若要返回正常的*MoveC*行为，则必须用*Wrist45*、*Wrist46*和*Wrist56*之外的一个旗标（比如*PathFrame*）来设置*CirPathMode*。



注意

由于仅使用了两根轴，因此在腕移动期间，位于表面之上的TCP高度将不可避免地起伏不定。这种高度起伏将取决于机器人位置、工具定义和圆弧半径等等。半径越大，高度波动就越大。考虑到这种高度波动，这里建议一开始以极低速度来执行这种移动，以此验证高度波动是否会变得过大，否则就可能导致切割工具与被切割表面相撞。

限制

如果存在以下情况，则无法使用*Wrist Move*选项：

- 工件正在移动
- 机器人被安装在导轨上或其它正在移动的机械臂上

只有运行*QuickMove*的第二代机器人才支持*Wrist Move*选项。

切割时该工具便不会与相关表面保持正确夹角，所以用这种方法切割出的孔洞将稍微成圆锥形。这对薄板来说通常无碍，但厚板就可能呈现出明显的锥度。

在切割期间，位于表面之上的TCP高度将起伏不定。待切割形状的尺寸越大，这种高度波动就越大，从而限制了相关形状的潜在大小，进而——除碰撞风险外——又限制了激光束焦距或水射流等工艺特点。

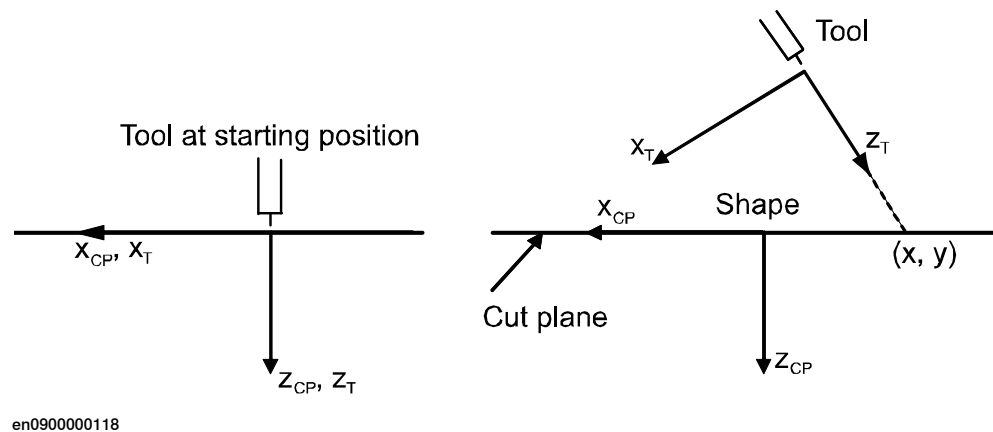
3.5.2 切割面框架

定义相关切割面框架

腕移动概念的关键在于切割面框架的定义（这种框架会提供对象表面的位置与方位信息）。在执行一条MoveC指令时，系统会按机器人的启动位置来定义该切割面框架，且该框架将被定义为“等同于启动位置处的工具框架”。注意对MoveC指令序列而言，整个序列中的切割面框架将始终都是相同的。

切割面插图

左图展示了如何定义相关切割面，右图则展示了切割期间的工具框架和切割面框架。



操作前提

鉴于定义切割面框架的方式，用户必须让启动位置满足以下条件：

- 该工具必须与相关表面保持正确夹角
- 该工具的z轴必须与激光束或水射流相互重合
- TCP必须尽量靠近相关表面

如果未满足头两项要求，那么切割轮廓的形状就会受到影响，比如让一个圆形孔洞更像是椭圆形。TCP往往会被定义在水射流喷口等的前方数毫米处，因此第三项要求通常易于满足；不过若未满足第三项要求，则只会影响到所得圆弧的半径，即是说切割弧的半径不符合编程半径（若是直线段，则会影响到其长度）。



提示

在FlexPendant示教器的点动窗口中，有一个按钮能让相关工具自动对准选定坐标框架。在开始腕移动时，用户可用此功能来确保相关工具与相关表面之间的夹角无误。



提示

腕移动不仅仅限于圆弧：如果MoveC的各目标点同为一轴，那么就能做出一条直线。

3 Motion performance

3.5.3 RAPID组件

3.5.3 RAPID组件

指令

此处简述了Wrist Move中的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型介绍中的相关描述。

指令	描述
CirPathMode	CirPathMode使用户能选择不同模式来重定圆形移动期间的工具方位。 自变数Wrist45、Wrist46和Wrist56专供Wrist Move选项使用。

3.5.4 RAPID代码示例

基本示例

此例展示了如何现用轴4和轴5、再用轴5和轴6做出两条圆弧。在做好两条圆弧后，CirPathMode便会停用腕移动。

```
! This position will define the cut plane frame
MoveJ p10, v100, fine, tWaterJet;

CirPathMode \Wrist45;
MoveC p20, p30, v50, z0, tWaterJet;

! The cut-plane frame remains the same in a sequence of MoveC
CirPathMode \Wrist56;
MoveC p40, p50, v50, fine, tWaterJet;

! Deactivate Wrist Movement, could use \ObjectFrame or \CirPointOri
  as well
CirPathMode \PathFrame;
```

高级示例

此例展示了如何用终点半径 R 和长度 $L+2R$ 在腕移动中切出一道槽。具体请参见第152页的插图，pSlot和wSlot。该槽的起点和终点都在pSlot这一位置（左半圆的中心）。为了避免在机器人中引发振荡，系统会沿着半圆引入路径和半圆引出路径——两者与槽的轮廓平顺相连——来开始和结束切割过程。给出的所有坐标都是相对于工件wSlot的坐标。

```
! Set the dimensions of the slot
R := 5;
L := 30;

! This position defines the cut plane frame, it must be normal to
  the surface
MoveJ pSlot, v100, z1, tLaser, \wobj := wSlot;
CirPathMode \Wrist45;

! Lead-in curve
MoveC Offs(pSlot, R/2, R/2, 0), Offs(pSlot, 0, R, 0), v50, z0,
  tLaser, \wobj := wSlot;

! Left semi-circle
MoveC Offs(pSlot, -R, 0, 0), Offs(pSlot, 0, -R, 0), v50, z0, tLaser,
  \wobj := wSlot;

! Lower straight line, circle point passes through the mid-point
  of the line
MoveC Offs(pSlot, L/2, -R, 0), Offs(pSlot, L, -R, 0), v50, z0,
  tLaser, \wobj := wSlot;

! Right semi-circle
MoveC Offs(pSlot, L+R, 0, 0), Offs(pSlot, L, R, 0), v50, z0, tLaser,
  \wobj := wSlot;
```

下一页继续

3 Motion performance

3.5.4 RAPID代码示例

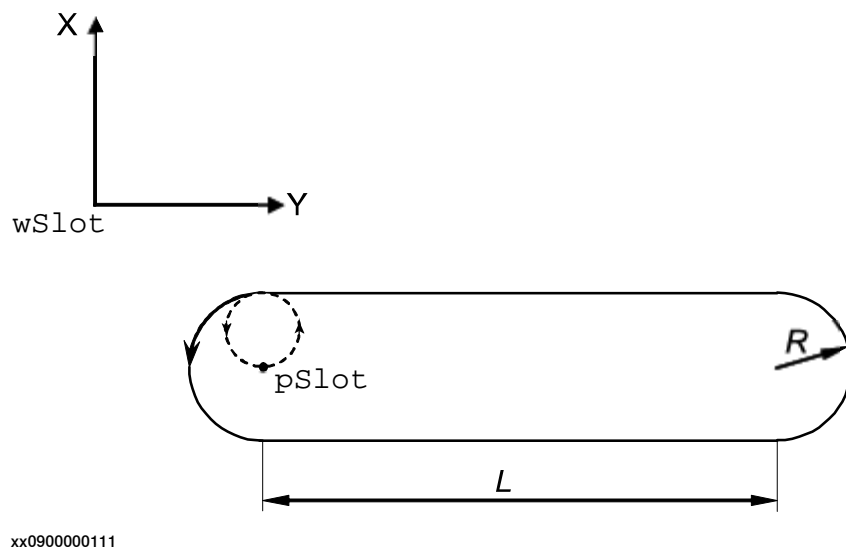
续前页

```
! Upper straight line, circle point passes through the mid-point
  of the line
MoveC Offs(pSlot, L/2, R, 0), Offs(pSlot, 0, R, 0), v50, z0, tLaser,
  \wobj := wSlot;

! Lead-out curve back to the starting point
MoveC Offs(pSlot, -R/2, R/2, 0), pSlot, v50, z1, tLaser, \wobj :=
  wSlot;

Deactivate Wrist Movement
CirPathMode \ObjectFrame;
```

插图, pSlot和wSlot



3.5.5 故障排查

意料之外的切割形状

如果得到的切割形状出乎意料，那么就检查以下事项：

- 该工具的z轴与激光束或水射流相互重合
- 在首次MoveC的启动位置处，该工具的z轴与相关表面之间的夹角无误
- 如果您拥有选项Advanced Shape Tuning，则请尝试对所涉腕轴进行摩擦微调。

错配半径

如果圆弧半径不符合编程半径，那么就检查启动位置处的TCP是否尽量靠近了相关表面。

无法靠选定的轴对移动

如果无法用选定轴对移动，那么就尝试用旗标Wrist45、Wrist46或Wrist56之一来激活另一轴对。若实在万不得已，则可尝试用另一套机器人配置来抵达启动位置。

此页刻意留白

4 Motion coordination

4.1 Machine Synchronization [607-1], [607-2]

4.1.1 概述

两种选项

*Machine Synchronization*有两个选项——*Sensor Synchronization*和*Analog Synchronization*组成。这两种选项的功能十分相似，只是硬件和配置有别。

两种选项之间的差别在于：

- *Analog Synchronization*要与一个相应的传感器（该传感器会以模拟信号的形式显示外部机械单元的位置）搭配使用。
- *Sensor Synchronization*需要一个编码器来计数外部机械单元移动时的脉冲，此外还需要一个编码器接口单元来把这些脉冲转变为传感器位置。

除非另有说明，否则本章的所有信息都是同时针对两种选项的。术语同步选项就是指这两种选项。如果某些信息仅对一种选项有效，那么将会说明其针对的是*Sensor Synchronization*还是*Analog Synchronization*。

目的

在一个传感器的帮助下，该同步选项会按一件外部移动装置（比如一台冲压机或传送器）来调节相应的机器人速度，此外也可用该选项来使两台机器人彼此同步。

描述

对同步而言，传感器的作用是检测压门、传送器、转台或类似装置的移动情况。用户可按该传感器的输出来调节机器人TCP的速度，因此在机器人抵达其编程目标点的同时，相应的外部装置也会抵达该装置自身的编程位置。

与外部装置间的同步不会影响到机器人TCP的路径，但会影响到机器人沿该路径移动时的速度。

功能

机器人控制器无法控制与该传感器相连的外部装置，不过这些装置与机器人控制器控制的机械单元间存在一些类似之处：

- 传感器位置会出现在FlexPendant示教器的点动窗口中
- 在执行MODPOS操作时，`robtarg`中会出现相应的传感器位置
- 可激活和停用的机械单元

基本方法

这是设置该同步选项的一般方式。更详细的具体做法请参见相应章节。

- 安装并连接硬件。
- 安装该同步软件。
- 配置相应的系统参数。
- 写入一段关联到相关传感器、可实现机器人同步移动的程序（或一段针对主动 / 从动机器人应用的程序）。

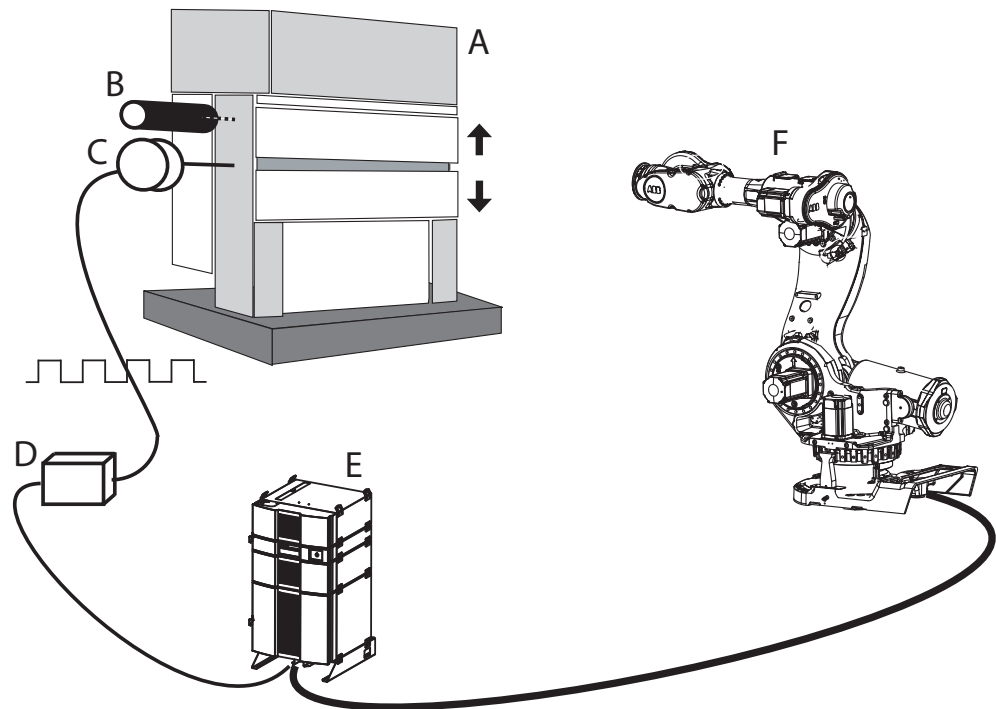
4 Motion coordination

4.1.2 所需事项

4.1.2 所需事项

Sensor Synchronisation

Sensor Synchronization应用由以下部分组成：

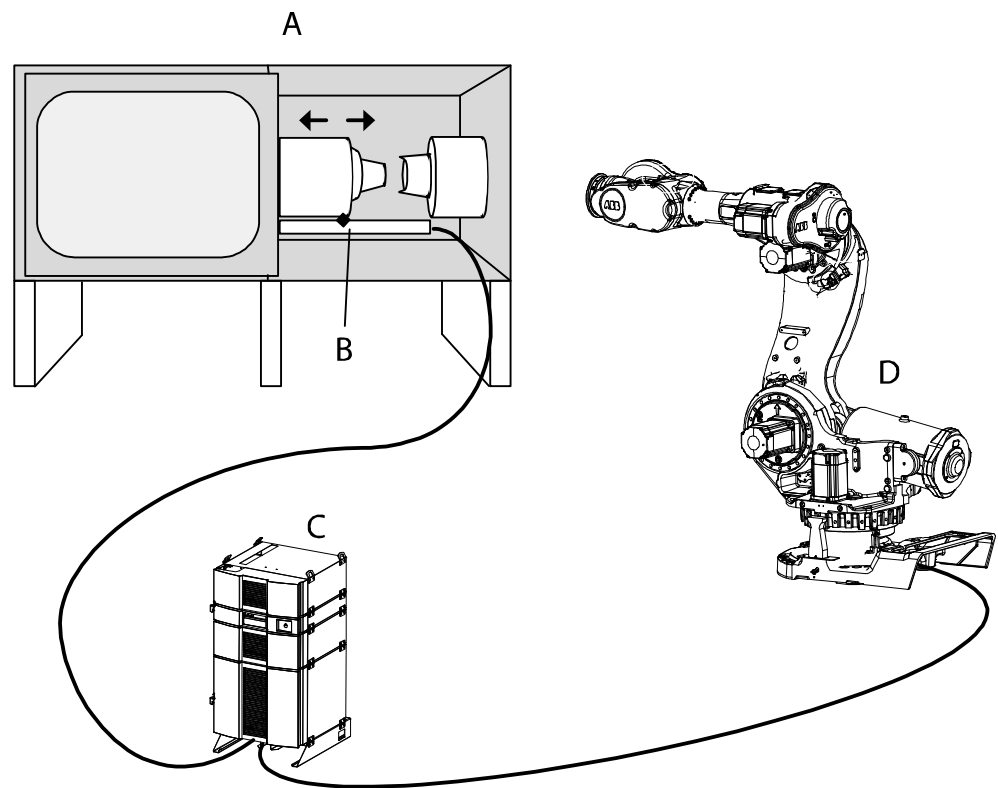


en0400000655

A	掌控机器人速度的外部装置（比如一道压门）
B	同步开关
C	编码器
D	编码器接口单元（DSQC 377）
E	控制器
F	机器人
B+C+D	作为一个传感器来为控制器提供输入信息

Analog Synchronization

Analog Synchronization应用由以下部分组成：



xx0700000431

A	掌控机器人速度的压模机
B	冲压机位置的模拟传感器
C	控制器
D	机器人

4 Motion coordination

4.1.3 同步特性

4.1.3 同步特性

功能

该同步选项提供了以下特性：

功能	描述
精确度	<p>在以恒定不变的传感器速度自动运行时，机器人的工具中心点（TCP）会停留在传感器所对应的编程位置内，且误差裕量为：</p> <ul style="list-style-type: none">• Sensor Synchronization为 + / - 50毫秒• Analog Synchronization为 + / - 100毫秒 <p>只要机器人还处在添加了传感器运动后的动态限制范围内，该特性就始终有效。该特性取决于机器人和传感器的校准情况，并仅适用于线性同步。</p>
对象队列	<p>仅用于Sensor Synchronization：</p> <p>外部装置每触发一次同步开关，系统就会在该对象队列中创建一个传感器对象。尽管Sensor Synchronization的这一队列通常不会包含一个以上的对象，但编码器接口单元还是会保持这一对象队列。</p>
传感器数据的RAPID权限	<p>RAPID程序有权通过相应传感器访问外部装置的当前位置和速度。</p>
多个传感器	<p>最多支持2个传感器。</p> <p>对Sensor Synchronization来说，每个传感器都必须有一个DSQC 377。</p>

4.1.4 对同步进程的一般性描述

以冲压机为例

此例展示了在搬运冲压机材料时使用同步功能的最基本步骤。

当...	那么.....
该冲压机已关闭，并做好了启动准备	一个来自机器人控制器（或PLC）的信号命令该冲压机启动。
该冲压机开始打开	对Sensor Synchronization而言，这会触发同步开关，并在对象队列中创建一个传感器对象。系统会把机器人与该对象关联起来。 对Sensor Synchronization和Analog Synchronization而言，机器人会在与冲压机同步的情况下朝冲压机移动，并在冲压机打开到足够大时抵达冲压机。
冲压机打开到足以让机器人进入的程度	机器人会在冲压机中放置（或移动）一个工件。同步结束。 对Sensor Synchronization而言，该传感器对象随后将被丢弃（即从对象队列中移除）。

4 Motion coordination

4.1.5 限制

4.1.5 限制

附加轴的限制

每个传感器都被视为一根附加轴，因此必须从“处于激活状态的6根附加轴”这一系统限值中减去已激活和已安装的传感器的数目。

安装的第一个传感器将使用测量节点6，第二个传感器将使用测量节点5。附加轴无法使用这些测量节点，且不宜在任何附加轴测量板的此类节点上连接任何旋转变压器。

在热重启或上电失败时丢失对象队列

仅用于Sensor Synchronization：

相关编码器接口单元（DSQC 377）会保留该对象队列。如果重启了系统，又或相关控制器或编码器接口单元电源失电，那么就会丢失该对象队列。

最小速度

为了保持运动的平顺和准确，所检测到的外部装置会有一个最小速度。如果该装置的移动速度慢于该最小速度，那么系统将把该装置视为静止。这一速度取决于所选的编码器，其范围可从4毫米 / 秒到8毫米 / 秒不等。

最大速度

该外部装置未预定任何最大速度。当各种速度超过对应的指定速度后，准确度便会降低，而机器人也无法在极高的传感器速度（>1000毫米 / 秒）下继续跟随相关传感器或继续保持在机器人动态限值内。

与选项Conveyor Tracking之间的兼容性

如果既安装了Machine Synchronization选项，又安装了Conveyor Tracking选项，那么同一时间只宜激活SSYNC1和CNV2的其中之一。

至于Machine Synchronization（Sensor Synchronization或Analog Synchronization），则必须停用CNV2。

至于Conveyor Tracking，则必须停用SSYNC1。

4.1.6 Sensor Synchronization的硬件安装

4.1.6.1 编码器规范

双相型

为了启用传感器反向运动的登记功能，同时也为了避免在传感器停止移动时因振动而产生错误计数，用于正交脉冲的编码器必须是双相型的编码器。

技术数据

输出信号：	打开集电极PNP输出端
电压：	10到30 V（通常由编码器接口单元的24 VDC提供）
电流：	50 - 100 mA
相位：	相移为90度的2相
工作周期：	50%
最大频率：	20千赫兹

编码器示例

符合这些标准的编码器示例为*Lenord & Bauer GEL 262*。

4 Motion coordination

4.1.6.2 对编码器的描述

4.1.6.2 对编码器的描述

概述

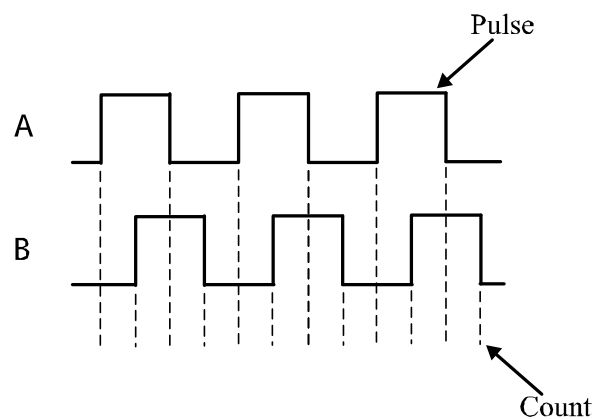
该编码器提供了一系列脉冲来指示传感器监测到的运动，其作用是实现机器人和相应外部装置间的运动同步。

脉冲通道

该编码器有两条脉冲通道A和B，两者的相位相差90°。每当旋转时，这两条通道便会分别发送固定数目的脉冲（具体取决于编码器的构造）。

- 由于关系到移动装置间的齿轮减速，因此用户必须选择该编码器每次旋转时的脉冲数目。
- 该编码器的脉冲比宜1250到2500道脉冲之间（传感器每运动一米时的脉冲数目）。
- 以正交方式使用通道A和B的脉冲，以便将脉冲比乘以四后得出计数值。

对脉冲比为1250到2500的编码器而言，这意味着相关控制软件将每米测量5000到10000次计数。



en0300000556

同步

为了能准确同步，该外部装置的移动范围不得超出某些限值（相对于机器人移动的限值）。若机器人每移动一米，该外部装置就必须移动0.2到5米（或弧度）。

4.1.6.3 安装建议

概述

安装该编码器时，必须要能精确反馈相关传感器的输出项（以反映相关外部装置的真实运动），这意味着最好把该编码器安装在尽量靠近机器人的位置，最远不超过30米。该编码器通常安装在相关外部装置的驱动单元上。可直接或通过齿轮皮带把该编码器连接在相应驱动单元的输出轴上。



注意

该编码器是一种敏感的测量装置，出于这一缘故，用户要注意两个要点：不能让除轴旋转以外的力从传感器传递到该编码器上；用减震器来安装该编码器，以防其因振动而受损。

放置

启动前要先考虑以下事项

如果	那么.....
该驱动单元包括了一种离合器布局	该编码器必须与离合器的传感器侧相连。
该编码器与一根驱动单元轴直接相连	很重要的一点就是安装一种专门设计的柔性联轴器，以免使该编码器的转子承受各种机械力。
相应外部装置的驱动单元远离该编码器	由于移动装置会在从驱动单元到编码器围笼的距离上伸缩，所以移动装置本身就是不准确度的一个来源。此时最好采用另一种联轴器布局来把编码器安装在更靠近驱动单元的位置处。

4 Motion coordination

4.1.6.4 连接编码器和编码器接口单元

4.1.6.4 连接编码器和编码器接口单元

概述

如果从机器人到编码器的电缆过长，那么就会因该电缆的电感而在编码器信号中产生尖峰脉冲。经过一段时间后，这种信号便会损坏编码器接口单元中的光电耦合器。
连接编码器接口单元方面详细信息请参见产品手册 - *IRC5*。

降低噪声

若要降低噪声，则用一根屏蔽电缆来连接相应的编码器。

减少尖峰脉冲

若要减少尖峰脉冲，则在信号电线与双相中每一相的接地之间安装一个电容器。可通过查看示波器上的编码器信号来确定正确的电容值。

电容器：

- 最好与连接有该编码器的端子板相连。
- 数值为100 nF到1 μ F，具体取决于相关电缆的长度。

编码器电源

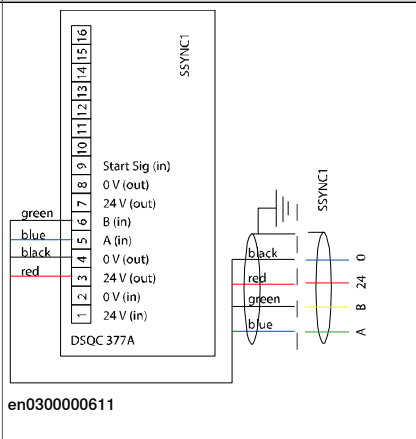
通常由编码器接口单元为该编码器提供24 VDC的电源。

若要在同一编码器上连接两个编码器接口单元，则请仅用一个编码器接口单元为该编码器供电。如果两个编码器接口单元都要供电，那么就必须在每个24 V DC连接器上都安装一个二极管，从而确保它们不会相互干扰。

连接编码器和同步开关

以下无返回值程序描述了如何把编码器和同步开关安装到相应的编码器接口单元上。

- 一个编码器可以与若干编码器接口单元相连。
- 如果有多台机器人要使用该传感器，那么就必须为每个控制器配备一个编码器接口单元。

操作	图示
1 把编码器连接到控制器的相应编码器接口单元 (DSQC 377) 上。	 en0300000611
2 把同步开关连接到控制器的相应编码器接口单元 (DSQC 377) 上。	

下一页继续

找出编码器旋转方向

以下无返回值程序描述了如何找出该编码器的旋转方向。

操作	图示
1 在FlexPendant示教器上轻击输入与输出。	
2 轻击视图，然后选择I / O单元	
3 向下卷动，然后选择Qtrack – d377	
4 向下卷动到c1位置	
5 在检查C1位置的同时向前执行该编码器。 如果其数目为递增： • 无需采取任何行动。 如果其数目为递减： • 位于编码器两面（0°和90°）的连接器必须能够互换。	<p>en0300000584</p>

4 Motion coordination

4.1.7.1 所需硬件

4.1.7 Analog Synchronization的硬件安装

4.1.7.1 所需硬件

模拟输入板

需要一块DSQC355A之类的模拟输入板。具体请参见应用手册 - *DeviceNet Master/Slave*。

模拟线性传感器

需要一个模拟线性传感器，且其模拟信号输入要在0到10 V之间。

4.1.8 软件安装

4.1.8.1 传感器的安装

概述

ABB公司通常会预先载入相应的同步选项和DeviceNet选项，因此无需重新安装这些选项。要想进一步了解如何在系统中添加选项，则请参见操作员手册 - *RobotStudio*。该同步选项会在系统参数中自动安装一个传感器。要想添加多个传感器，则请参见[第170页的多个传感器的安装](#)。

关于安装

这些选项将安装三种附加配置：

- 相关编码器接口单元的I / O（仅用于Sensor Synchronization）
- 传感器进程描述
- 运动机械描述

配置Sensor Synchronization的默认安装项

该无返回值程序描述了如何在RobotStudio的配置编辑器中配置Sensor Synchronization的系统参数。

	操作
1	将该单元的参数 <i>Connected to Bus</i> 从“Virtual1”更改为正确的总线（比如“DeviceNet1”）。
2	指定该单元的正确地址（参数 <i>DeviceNet Address</i> ）。
3	如果更改了参数 <i>DeviceNet Master Address</i> （在主题I/O下的类型 <i>Bus</i> 中），那么就必须把实例 <i>TimeKeeperInit</i> 的参数 <i>Default Value</i> （在主题I/O下的类型 <i>Fieldbus Command Type</i> 中）更改为同一数值。

配置Analog Synchronization的默认安装项

该无返回值程序描述了如何在RobotStudio的配置编辑器中配置Analog Synchronization的系统参数。

	操作
1	将该单元的单元类型参数 <i>Type of Unit</i> 从“Virtual”更改为正确的单元类型（比如“d355A”）。
2	将该单元的参数 <i>Connected to Bus</i> 从“Virtual1”更改为正确的总线（比如“DeviceNet1”）。
3	指定该单元的正确地址（参数 <i>DeviceNet Address</i> ）。
4	将相关单元类型（如d355A）的通信间隔时间从50毫秒改为20毫秒（参数 <i>Connection 1 Interval</i> ）。 该参数方面的更多信息请参见应用手册 - <i>DeviceNet Master/Slave</i> 。

如何为Sensor Synchronization手动添加一个传感器

使用以下无返回值程序来手动添加一个传感器。

	操作
1	将编码器接口单元与相应的CAN总线相连。请注意该CAN总线上的地址。
2	在RobotStudio中点击载入参数。
3	选择：若无相同参数则载入参数，然后点击打开。

下一页继续

4 Motion coordination

4.1.8.1 传感器的安装

续前页

	操作
4	安装与DeviceNet1（第一块板）相连的一个主动传感器。 从选项 / CNV目录中逐一载入下列文件： <ul style="list-style-type: none">• <i>syvm1_eio.cfg</i>• <i>syvm1_prc.cfg</i>• <i>syvm1_moc.cfg</i>
5	安装与DeviceNet2（第二块板）相连的一个从动传感器。 从选项 / CNV目录中逐一载入下列文件： <ul style="list-style-type: none">• <i>syvs1_eio.cfg</i>• <i>syvs1_prc.cfg</i>• <i>syvs1_moc.cfg</i>
6	重启系统。
7	必要时就校正新编码器接口单元的地址。最好用相关单板的实际地址来取代文件 <i>syvxx_eio.cfg</i> 中的默认地址。

如何为Analog Synchronization手动添加一个传感器

没有为Analog Synchronization添加一个传感器的预定义文件，不过用户可以复制下列文件，然后根据第二个传感器来编辑它们：

- *synvaileio.cfg*
- *synvailprc.cfg*
- *syim1.moc*

4.1.8.2 重新载入保存的Motion参数

概述

在安装同步选项时，系统将在Motion系统参数中载入各附加轴的专用传感器配置。



注意

如果在相关同步选项前载入这些参数，那么机械单元SSYNC1就不会出现在点动窗口下的FlexPendant示教器上。

重新载入该SSYNC1参数

使用RobotStudio，然后遵守下列步骤（更多信息请参见操作员手册 - *RobotStudio*）：

	操作
1	打开“配置编辑器”，然后选择主题 <i>Motion</i> 。
2	选择类型 <i>File</i> 。
3	点击载入参数，然后选择模式。
4	点击打开，然后从RobotWare安装包中选择文件syn1_moc。
5	重启控制器，从而使所做改动生效。

结果

此时点动窗口中的FlexPendant示教器就理应能使用机械单元SSYNC1了。

4 Motion coordination

4.1.8.3 多个传感器的安装

4.1.8.3 多个传感器的安装

关于安装

ABB公司通常会预先载入相应的同步选项和DeviceNet选项，因此无需重新安装这些选项。要想进一步了解如何在系统中添加选项，则请参见操作员手册 - *RobotStudio*。该同步选项会在系统参数中自动安装一个传感器。

DeviceNet Dual选项

若包括了DeviceNet Dual，则会在系统中安装以下三个传感器：

- 一个“‘机器人到冲压机之间的同步’类型”的传感器：SSYNC1
- 一个虚拟的主动传感器：SSYNM1
- 一个虚拟的从动传感器：SSYNCS1

手动添加传感器

同一控制器最多能使用四个传感器，但用户必须手动载入其中三个额外传感器的相关参数。

使用以下无返回值程序来手动载入这些传感器。

	操作
1	至于Sensor Synchronization，则将编码器接口单元与相应的CAN总线相连。请注意该CAN总线上的地址。
2	用RobotStudio来添加新的参数。
3	点击载入参数。
4	选择：若无相同参数则载入参数，然后点击打开。
5	安装与DeviceNet1（第一块板）相连的一个主动传感器。 从选项 / CNV目录中逐一载入下列文件： <ul style="list-style-type: none">• 对于第二个传感器：syvm2_eio.cfg、syvm2_prc和syvm2_moc.cfg• 对于第三个传感器：syvm3_eio.cfg、syvm3_prc.cfg和syvm3_moc.cfg• 对于第四个传感器：syvm4_eio.cfg、syvm4_prc.cfg和syvm4_moc.cfg
6	安装与DeviceNet2（第二块板）相连的一个从动传感器。 从选项 / CNV目录中逐一载入下列文件： <ul style="list-style-type: none">• 对于第二个传感器：syvs2_eio.cfg、syvs2_prc.cfg和syvs2_moc.cfg• 对于第三个传感器：syvs3_eio.cfg、syvs3_prc.cfg和syvs3_moc.cfg• 对于第四个传感器：syvs4_eio.cfg、syvs4_prc.cfg和syvs4_moc.cfg
7	重启系统。
8	对于传感器同步：必要时就校正新编码器接口单元的地址。在主题I/O下的系统参数中找出各自的编码器接口单元。最好用相关单板的实际地址来取代文件syvxx_eio.cfg中的默认地址。

可用的传感器

此时第二和第三个传感器（SSYNC2和SSYNC3）理应出现在*Motion/mechanical unit*中和FlexPendant示教器上的点动窗口中。

4.1.9 同步编程

4.1.9.1 用同步选项编程时的一般问题

激活传感器

必须先激活该传感器，然后该传感器才能像其它所有机械单元那样用于工件协调。用平常的ActUnit指令来激活该传感器，用DeactUnit来停用该传感器。

安装该传感器时默认启动时不激活该传感器。若有需要，也可将该传感器配置成“始终在启动时激活”。具体请参见第204页的机械单元。

自动连接

仅用于Sensor Synchronization：

在激活一个传感器机械单元时，首先检查相应编码器接口单元的状态（以便查看之前是否连接过该传感器）。如果相应编码器接口单元通过I / O信号c1已连接来显示连接，则系统将在激活时自动连接该传感器。这种特性的作用是在上电失败时用相应编码器接口单元上的后备电源自动重连。

通过WaitSensor指令相连

除非已用一条WaitSensor指令将一个对象与该传感器相连，否则无法编写那些将与外部装置同步的运动。

如果上一条WaitSensor指令已连接了该对象，或曾在激活时建立了连接，那么执行第二条WaitSensor指令将会导致一项错误。

在用一条WaitSensor指令与一个对象相连后，使用SyncToSensor\On指令来启动已同步的相关运动。

指令WaitSensor和SyncToSensor\On方面的更多细节请参见技术参考手册 - RAPID指令、函数和数据类型。

编写Sensor Synchronization

以下指令引用了一些编程示例。

	操作	Information
1	用下列指令创建一段程序： ActUnit SSYNCl; MoveL waitp, v1000, fine, tool; WaitSensor SSYNCl;	
2	单步执行越过了WaitSensor指令的该程序。	如果对象队列中有一个对象，那么就会返回该指令；如果没有对象，那么就会停止执行并等候一个对象（比如一个同步信号）。
3	运行相关外部装置，直至同步开关生成一个同步信号位置。	该程序理应退出了WaitSensor，并在此时与该对象“相关联”。
4	在您打算编程的机器人目标点理应对应的位置处停止该外部装置。	
5	用一条SyncToSensor SSYNCl\On指令启动同步运动。具体请参见第173页的编程示例。	

下一页继续

4 Motion coordination

4.1.9.1 用同步选项编程时的一般问题

续前页

	操作	Information
6	程序移动指令。 每当您修改了一个位置时，就请将相关外部装置执行到机器人目标点所理应对应的位置处。	使用相关移动指令的角区。具体请参见第177页的精确点编程。
7	用一条SyncToSensor SSYNCl\Off指令结束同步运动。具体请参见第173页的编程示例。	
8	仅用于Sensor Synchronization : 编写一条DropSensor SSYNCl;指令。具体请参见第173页的编程示例。	
9	如果这是程序末尾，或如果不再需要相关传感器，那么就编写一条DeactUnit SSYNCl;指令。具体请参见第173页的编程示例。	

使传感器同步

如果无法将相关外部装置移动到所需位置，那么就先修改该位置，然后编辑机器人目标点中的相应传感器数值（就像任何附加轴那样）。

4.1.9.2 编程示例

Sensor Synchronization程序

```
MoveJ p0, vmax, fine, tool1;

!Activate sensor
ActUnit SSYNC1;

!Connect to the object
WaitSensor SSYNC1;

!Start the Synchronized motion
SyncToSensor SSYNC1\On;

!Instructions with coordinated robot targets
MoveL p10, v1000, z20, tool1;
MoveL p20, v1000, z20, tool1;
MoveL p30, v1000, z20, tool1;

!Stop the synchronized motion
SyncToSensor SSYNC1\Off;

!Exit coordinated motion
MoveL p40, v1000, fine, tool1;

!Disconnect from current object
DropSensor SSYNC1;

MoveL p0, v1000, fine;

!Deactivate sensor
DeactUnit SSYNC1;
```

Analog Synchronization程序

```
VAR num startdist := 600;

MoveJ p0, vmax, fine, tool1;

!Activate sensor
ActUnit SSYNC1;

WaitSensor SSYNC1 \RelDist:=startdist;

!Start the Synchronized motion
SyncToSensor SSYNC1\On;

!Instructions with coordinated robot targets
MoveL p10, v1000, z20, tool1;
MoveL p20, v1000, z20, tool1;
MoveL p30, v1000, z20, tool1;
```

下一页继续

4 Motion coordination

4.1.9.2 编程示例

续前页

```
!Exit coordinated motion
MoveL p40, v1000, fine, tool1;

!Stop the synchronized motion
SyncToSensor SSYNC1\Off;

MoveL p0, v1000, fine;

!Deactivate sensor
DeactUnit SSYNC1;
```

4.1.9.3 进入和退出角区中的协调运动

可以使用角区

一旦将WaitSensor指令与一个对象关联起来，就可可通过角区进入和退出与传感器相互同步的运动。

丢弃角区后的对象

如果使用一条采用角区的指令来退出协调运动，那么就无法直接续接DropSensor指令，从而导致系统就在机器人离开角区前就丢弃了相关对象，而此时的相关运动却仍需这一传送器协调工件。

如果在相关运动仍需要工件位置时丢弃了该工件，那么就会发生停止。

为避免发生这种情况，请要么调用一条精确点指令，要么在丢弃相关工件前至少使用两条角区指令。

正确示例

此例展示了如何通过角区来进入和退出协调运动。

```
MoveL p10, v1000, fine, tool1;  
WaitSensor SSYNC1;  
MoveL p20, v500, z50, tool1;  
!start synchronization after zone around p20  
SyncToSensor SSYNC1\On  
MoveL p30, v500, z20, tool1;  
MoveL p40, v500, z20, tool1;  
MoveL p50, v500, z20, tool1;  
MoveL p60, v500, z50, tool1;  
!Exit synchronization after zone around p60  
SyncToSensor SSYNC1\Off;  
MoveL p70, v500, fine, tool1;  
DropSensor SSYNC1;  
MoveL p10, v500, fine, tool1;
```

错误示例

这是退出角区协调的错误示例，所示做法将导致相关程序随着一项错误而停止。

```
MoveL p50, v500, z20, tool1;  
MoveL p60, v500, z50, tool1;  
!Exit coordination in zone  
SyncToSensor SSYNC1\Off;  
DropSensor SSYNC1;
```

如果角区中的协调运动结束，那么就必须在丢弃传感器前执行令一条移动指令。

4 Motion coordination

4.1.9.4 使用若干传感器

4.1.9.4 使用若干传感器

概述

若使用了若干个传感器，相关程序就必须至少有一条移动指令，且与两个不同传感器同步的路径各部分之间不得形成任何同步。

程序示例

```
!Connect to the object
WaitSensor SSYNC1\RelDist:=Pickdist;

!Start the Synchronized motion
SyncToSensor SSYNC1\MaxSync:=1653\On;

!Instructions with coordinated robot targets
MoveL p30, v400, z20, currtool;

!Stop the synchronized motion
SyncToSensor SSYNC1\Off;

!Instructions with coordinated robot targets
MoveL p31, v400, z20, currtool;

!Connect to the object
WaitSensor SSYNC2\RelDist:=1720;

!Instructions with coordinated robot targets
MoveL p32, v400, z50, currtool;

!Start the Synchronized motion
SyncToSensor SSYNC2\MaxSync:=2090\On;

!Instructions with coordinated robot targets
MoveL p33, v400, z20, currtool;

!Stop the synchronized motion
SyncToSensor SSYNC2\Off;
```


4.1.9.5 精确点编程

概述

若采用了同步运动，则避免使用精确点。相关机器人将会停止，并与相关传感器丧失为期100毫秒的同步。随后系统会继续执行RAPID。

如果最后一个目标点不需要准确的同步，那么就可以对最后一条同步移动指令进行精确点编程。

程序示例

以下程序示例展示了可能出现的同步运动停止方式。

```
WaitSensor SSYNC1;  
SyncToSensor SSYNC1 \On;  
MoveL p1, v500, z20, tool1;  
MoveL p2, v500, fine, tool1;  
SyncToSensor SSYNC1 \Off;  
MoveL p3, v500, z20, tool1;  
MoveL p4, v500, fine, tool1;  
DropSensor SSYNC1;
```

p4处的机器人不再与相关外部装置同步，也没有使用精确点方面的限制。

相关同步会在p2处结束——此处可以使用一个精确点，但同步准确度将会降低。

4.1.9.6 丢弃传感器对象

概述

对Sensor Synchronization而言，一旦结束了同步运动，便可用DropSensor指令丢弃一个已关联的对象。

示例：`DropSensor SSYNCL;`

对于Analog Synchronization而言，用户不得使用指令DropSensor。

考虑因素

丢弃一个对象时必须考虑到下列因素：

- 很重要的一点就是确保当丢弃该对象时，相关机器人运动已不再使用相应的传感器位置。如果机器人运动仍需要该传感器位置，那么就会在丢弃该对象时发生一次停止。
- 只要还未发出SyncToSensor \Off指令，机器人运动就会与相关的传感器保持同步。
- 不必为了执行一条DropSensor指令而进行关联。即使没有已关联的对象，也不会返回任何错误。

4.1.9.7 FlexPendant示教器上的信息

概述

用户有权通过FlexPendant示教器访问相关传感器的位置和速度

点动窗口

点动窗口展示了相关传感器对象的位置（以毫米为单位）。如果定义了*Queue Tracking Distance*，那么其将为负值。在触发同步开关时，系统会自动更新点动窗口中的这一位置。

I / O窗口

Sensor Synchronization

用户有权从I / O窗口访问编码器接口单元上定义的所有信号，并可在该窗口中查看相应的传感器对象位置（以米为单位）和传感器对象速度（以米 / 秒为单位）。在相关同步开关登记一个传感器对象前，该速度将始终为0米 / 秒。

Analog Synchronization

对Analog Synchronization而言，I / O窗口只会显示相关的传感器位置。

4 Motion coordination

4.1.9.8 编写考虑因素

4.1.9.8 编写考虑因素

效果限值

如果达到了关节速度限制（尤其是在奇点中），则会丧失同步。程序员有责任确保同步移动期间的路径不会超出相关机器人的速度与运动能力。

运动命令

同步期间允许使用任何运动指令。

手动模式

手动模式下的同步处于未激活状态。

速度减少%按钮

该同步仅会在100%的速度下发挥作用。如果按传感器的移动来调节相关的机器人速度，那么系统就会对已定义的机器人速度百分数进行超驰。

编程速度

接近真实执行速度的编程速度能带来效果最好的同步。宜把最适当的执行速度选为编程速度。两条移动指令的速度不宜相差太大。

精确点

同步移动期间允许使用精确点，但机器人会在精确点处停止。若相关外部装置仍在移动，则会丧失相应同步。具体请参见[第177页的精确点编程](#)。

位置警告

如果`robot_to_sensor`位置比大于10或小于0.1，则会出现一条警告。用户宜按其警告文本来修改`robtarg`位置或`robtarg`中的传感器数值。

速度警告

如果编写的`sensor_speed`快于：

- $(\text{max_sync_speed} * \text{sensor_nominal_speed}) / \text{robot_tcp_speed}$

之后会出现一条速度警告，而用户则宜按其警告文本来修改机器人速度、`sensor_nominal_speed`或`max_sync_speed`。

如果编写的`sensor_speed`慢于：

- $(\text{min_sync_speed} * \text{sensor_nominal_speed}) / \text{robot_tcp_speed}$

将出现一条类似的警告：

- `Programmed_sensor_speed equals sensor_distance/robot_interpolation_time.`

更换工具

如果使用了`corvec`，则不允许在同步期间更换工具。

下一页继续

将停用相关同步的各条指令

指令ActUnit、DeactUnit和ClearPath将停用任何SyncToSensor或SupSyncSensorOn指令。所以不宜在SyncToSensor或SupSyncSensorOn指令与涉及同步路径或监控路径的移动指令之间使用指令ActUnit、DeactUnit和ClearPath。

正确的顺序为：

```
ActUnit SSYNCl;  
WaitSensor SSYNCl;  
SyncToSensor SSYNCl\On;  
! move instructions  
...  
SyncToSensor SSYNCl\Off;
```

其它RAPID限制

- 命令StorePath、RestoPath在同步期间不起作用。
- EoffsSet、EoffsOn和EoffsOff会影响传感器示教位置。
- 无法用同步选项实现上电失败重启。

4.1.9.9 运行模式

手动减速模式下的运行 (<250毫米 / 秒)

可用前进和后退硬按钮来对相关程序进行单步调试。用户既可以添加新指令，也可以用MODPOS来修改编程位置。

如果在运动期间放开了启用装置，相关机器人则会恢复如常。

当处于手动减速模式时，机器人会执行针对相关传感器的同步运动。

自动模式下的运行

一旦执行了一条SyncToSensor指令，就再也无法在传感器移动期间用前进和后退按钮来对相关程序进行单步调试。

启动 / 停止

如果按下了“停止”按钮，或在SyncToSensor与DropSensor指令之间执行了RAPID指令Stop或StopMove，那么相关机器人就会停止并解除同步。

该传感器对象不会丢失，不过若该传感器正在移动，那么其对象就会迅速移出最大距离。如果传感器正在移动，则不允许用当前指令来重启同步。必须从MAIN来重启该程序。如果强制重启，那么相关机器人将伴随着max_dist错误而停在传感器之前停止的位置。

紧急停止 / 重启

按下紧急停止时，相关机器人会立即停止。如果在SyncToSensor之后停止相关程序，那么虽不会丢失相应的传感器对象，但如果该传感器正在移动，那么该对象就会迅速移出最大距离。无法用当前指令来重启同步，所以必须用MAIN来重启该程序。如果在询问“您是否想重获”后强制重启，那么相关机器人将按编程速度、以非同步的方式移动到该传感器处。

手动全速模式 (100%) 下的运行

手动全速模式下的运行与自动模式下的运行相仿。用户可按住启动按钮来执行相关程序，但一旦执行了一条SyncToSensor指令，就再也无法在传感器移动期间用前进和后退按钮来对该程序进行单步调试。

“松手即停”按钮

若按下又放开“松手即停”按钮，则相关机器人会先停止然后重启。机器人停止时会丧失同步，而重启时机器人则会尝试重获max_adjustment_speed下的同步。

停止 / 重启

按下停止按钮或紧急停止时，相关机器人会立即停止。如果在SyncToSensor之后停止相关程序，那么虽不会丢失相应的传感器对象，但如果该传感器正在移动，那么该对象就会迅速移出最大距离。无法用当前指令来进行重启，所以必须用MAIN来重启该程序。

4.1.10 机器人与机器人之间的同步

4.1.10.1 简介

概述

可以在同一项同步应用中同步两套机器人系统。通过一套主动机器人设定和一套从动机器人设定来实现这种同步。

要求

电缆连接与设定方面的信息请参见应用手册 - *DeviceNet Master/Slave*。

4 Motion coordination

4.1.10.2 “机器人与机器人之间的同步”概念

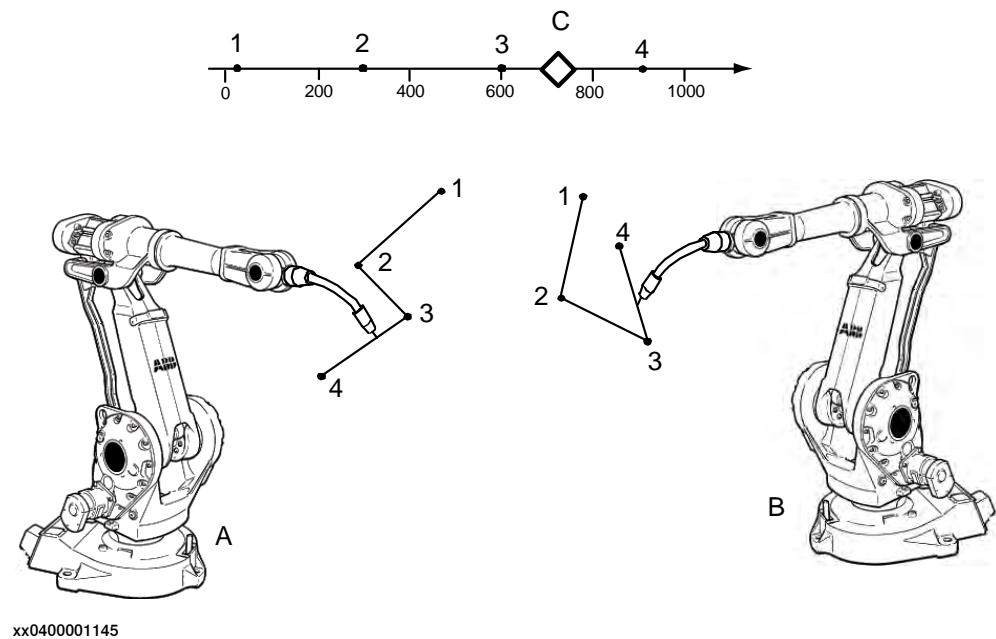
4.1.10.2 “机器人与机器人之间的同步”概念

描述

“机器人与机器人之间的同步”的基本思路是两台机器人宜使用共同的虚拟传感器。由主动机器人控制该传感器的虚拟运动，而从动机器人则按该传感器的虚拟位置和速度来调节自身速度。

若想实现同步，则定义两台机器人在同一时间宜处于的位置，并为每个点指定一个传感器数值。

图示



4.1.10.3 主动机器人的配置参数

概述

用下列参数来设置主动机器人。
用RobotStudio来更改这些参数。

主题：Motion

SINGLE_TYPE / 参数	值
<i>Name</i>	SSYNC2
<i>mechanics</i>	SS_LIN
<i>process_name</i>	SSYNC2
<i>use_path</i>	PSSYNC

主题：进程

SENSOR_SYSTEM / 参数	值
<i>Name</i>	SSYNC1
<i>sensor_type</i>	CAN
<i>use_sensor</i>	CAN1
<i>adjustment_speed</i>	1000
<i>min_dist</i>	600
<i>max_dist</i>	20000
<i>correction_vector_ramp_length</i>	10

主题：I/O

EIO_UNIT

EIO_UNIT / 参数	值
名称	MASTER1
<i>UnitType</i>	DN_SLAVE
<i>Bus</i>	DeviceNet1
<i>DN_Address</i>	1

EIO_SIGNAL

EIO_SIGNAL / 参数	值
<i>Name</i>	ao1Position
<i>SignalType</i>	AO
<i>Unit</i>	MASTER1
<i>UnitMap</i>	0-15
<i>MaxLog</i>	10.0
<i>MaxPhys</i>	1
<i>MaxPhysLimit</i>	1

[下一页继续](#)

4 Motion coordination

4.1.10.3 主动机器人的配置参数

续前页

EIO_SIGNAL / 参数	值
<i>MaxBitVal</i>	32767
<i>MinLog</i>	-10.0
<i>MinPhys</i>	-1
<i>MinPhysLimit</i>	-1
<i>MinBitVal</i>	-32767

EIO_SIGNAL / 参数	值
<i>Name</i>	ao1Speed
<i>SignalType</i>	AO
<i>Unit</i>	MASTER1
<i>UnitMap</i>	16-31
<i>MaxLog</i>	10.0
<i>MaxPhys</i>	1
<i>MaxPhysLimit</i>	1
<i>MaxBitVal</i>	32767
<i>MinLog</i>	-10.0
<i>MinPhys</i>	-1
<i>MinPhysLimit</i>	-1
<i>MinBitVal</i>	-32767

EIO_SIGNAL / 参数	值
<i>Name</i>	ao1PredTime
<i>SignalType</i>	AO
<i>Unit</i>	MASTER1
<i>UnitMap</i>	32-47
<i>MaxLog</i>	10.0
<i>MaxPhys</i>	1
<i>MaxPhysLimit</i>	1
<i>MaxBitVal</i>	32767
<i>MinLog</i>	-10.0
<i>MinPhys</i>	-1
<i>MinPhysLimit</i>	-1
<i>MinBitVal</i>	-32767

EIO_SIGNAL / 参数	值
<i>Name</i>	do1Dready
<i>SignalType</i>	DO
<i>Unit</i>	MASTER1
<i>UnitMap</i>	48

下一页继续

EIO_SIGNAL / 参数	值
<i>Name</i>	do1Sync2
<i>SignalType</i>	DO
<i>Unit</i>	MASTER1
<i>UnitMap</i>	50

4 Motion coordination

4.1.10.4 从动机器人的配置参数

4.1.10.4 从动机器人的配置参数

概述

默认配置方面的信息请参见[第203页的系统参数](#)。
用RobotStudio来更改相关参数和设置从动机器人。

描述

若要让从动机器人停止，并在与主动机器人同步的情况下重启前者，则：

- 将参数值`min_sync_speed`设置成0.0

如果在主动机器人路径中定义了一个精确点，那么从动机器人也会停止。

主题：Process

SENSOR_SYSTEM

SENSOR_SYSTEM / 参数	值
<i>Name</i>	SSYNCS1
<i>sensor_type</i>	CAN
<i>use_sensor</i>	CAN1
<i>adjustment_speed</i>	1000
<i>min_dist</i>	600
<i>max_dist</i>	20000
<i>correction_vector_ramp_length</i>	10
<i>nominal_speed</i>	1000

CAN_INTERFACE

CAN_INTERFACE / 参数	值
<i>Name</i>	CAN1
<i>Signal delay</i>	34
<i>Connected signal</i>	c1Connected
<i>Position signal</i>	c1Position
<i>Velocity signal</i>	c1Speed
<i>Null speed signal</i>	c1NullSpeed
<i>Data ready signal</i>	
<i>Waitwobj signal</i>	c1WaitWObj
<i>Dropwobj signal</i>	c1DropWObj
<i>Data Time stamp</i>	c1DTimestamp
<i>RemAllPObj signal</i>	c1RemAllPObj
<i>Virtual sensor</i>	NO
<i>Sensor Speed filter</i>	0,33

下一页继续

主题：I/O

EIO_UNIT

EIO_UNIT / 参数	值
<i>Name</i>	SLAVE1
<i>UnitType</i>	DN_SLAVE
<i>Bus</i>	DeviceNet2
<i>DN_Address</i>	1

EIO_SIGNAL

EIO_SIGNAL / 参数	值
<i>Name</i>	ai1Position
<i>SignalType</i>	AI
<i>Unit</i>	SLAVE1
<i>UnitMap</i>	0-15
<i>MaxLog</i>	10.0
<i>MaxPhys</i>	1
<i>MaxPhysLimit</i>	1
<i>MaxBitVal</i>	32767
<i>MinLog</i>	-10.0
<i>MinPhys</i>	-1
<i>MinPhysLimit</i>	-1
<i>MinBitVal</i>	-32767

EIO_SIGNAL / 参数	值
<i>Name</i>	ai1Speed
<i>SignalType</i>	AI
<i>Unit</i>	SLAVE1
<i>UnitMap</i>	16-31
<i>MaxLog</i>	10.0
<i>MaxPhys</i>	1
<i>MaxPhysLimit</i>	1
<i>MaxBitVal</i>	32767
<i>MinLog</i>	-10.0
<i>MinPhys</i>	-1
<i>MinPhysLimit</i>	-1
<i>MinBitVal</i>	-32767

EIO_SIGNAL / 参数	值
<i>Name</i>	ai1PredTime
<i>SignalType</i>	AI

下一页继续

4 Motion coordination

4.1.10.4 从动机器人的配置参数

续前页

EIO_SIGNAL / 参数	值
<i>Unit</i>	SLAVE1
<i>UnitMap</i>	32-47
<i>MaxLog</i>	10.0
<i>MaxPhys</i>	1
<i>MaxPhysLimit</i>	1
<i>MaxBitVal</i>	32767
<i>MinLog</i>	-10.0
<i>MinPhys</i>	-1
<i>MinPhysLimit</i>	-1
<i>MinBitVal</i>	-32767

EIO_SIGNAL / 参数	值
<i>Name</i>	di1Dready
<i>SignalType</i>	DI
<i>Unit</i>	SLAVE1
<i>UnitMap</i>	48

EIO_SIGNAL / 参数	值
<i>Name</i>	di1Sync2
<i>SignalType</i>	DI
<i>Unit</i>	SLAVE1
<i>UnitMap</i>	50

4.1.10.5 主动机器人的编程示例

概述

以下程序示例展示了如何编写一台主动机器人的程序。

主动机器人编程

```
syncstart:=20;
Syncpos1:=300;
Syncpos2:=600;
Syncpos3:=900;
Syncpos4:=1200;

!Synchronized motion between master and slave
robpos1.extax.eax_e:=syncpos1;
robpos2.extax.eax_e:=syncpos2;
robpos3.extax.eax_e:=syncpos3;
robpos4.extax.eax_e:=syncpos4;
robpos5.extax.eax_e:=syncstart;

!Init of external axis
pOutsideNext.extax.eax_e:=syncstart;

!Activate sensor
ActUnit SSYNCl;

!Instruction with coordinated robot targets
MoveJ pOutsideNext, v1000, fine, tool1;

!Init of external axis
robposstart.extax.eax_e:=syncstart;

!Set digital output
SetDO Dosync 1,0

!Instructions with coordinated robot targets
MoveJ robposstart, v2000, z50, tool1;

!Set digital output
PulseDO\PLength:= 0.1, doSync1;

!Instructions with coordinated robot targets
MoveJ robpos1, v2000, z10, tool1;
MoveJ robpos2, v2000, z10, tool1;
MoveJ robpos3, v2000, z10, tool1;
MoveJ robpos4, v2000, z10, tool1;
MoveJ robpos5, v2000, z10, tool1;
```

下一页继续

考虑因素

要考虑到以下事项

- 每个机器人目标点在同步期间的`extax.eax_e`数值都宜有所增加。在同步后，主动机器人的第一条移动指令的`extax.eax_e`数值宜大于上一条指令的该数值，否则`extax.eax_e`数值可能会减小，并导致在从动机器人抵达其目标点前就结束了同步。
- 移回`syncstart`（本例中移至`robpos5`的移动指令）的速度可能慢于预定速度（`v2000`）如果此次机器人移动距离补偿，且`extax.eax_e`数值较大，那么最大速度就将受限于虚拟传感器的速度。
- 请勿使用`WaitSensor`或`DropSensor`。
- 验证虚拟传感器的最大速度（`speed_out`）是否慢于1米 / 秒。

4.1.10.6 从动机器人的编程示例

概述

以下程序示例展示了如何编写一台从动机器人的程序。

从动机器人编程

```
syncstart:=20;
Syncpos1:=300;
Syncpos2:=600;
Syncpos3:=900;

!Synchronized motion between master and slave
robpos1.extax.eax_e:=syncpos1;
robpos2.extax.eax_e:=syncpos2;
robpos3.extax.eax_e:=syncpos3;

!Instructions with coordinated robot targets
MoveJ posstart, v500, z50, tool1;

!Wait for digital input
WaitDI diSync1; 1;

!Connect to the object
WaitSensor SSYNCl;\RelDist:=100;

!Start the Synchronized motion
SyncToSensor SSYNCl\On;

!Instructions with coordinated robot targets
MoveJ robpos1, v2000, z10, tool1;
MoveJ robpos2, v2000, z10, tool1;
MoveJ robpos3, v2000, z10, tool1;

!Stop the synchronized motion
SyncToSensor SSYNCl\Off;
```

考虑因素

要考虑到以下事项：

- 请勿使用DropSensor。
- 请勿使用任何corvec。

4.1.11 用已记录的曲线来与液压式冲压机同步

4.1.11.1 介绍

概述

本节描述了如何用已记录的机器曲线来改善机器人与一台液压式冲压机之间的同步准确度。该曲线的作用是建立压制路径模型。若不使用已记录的曲线，则机器人与冲压机模型之间需要有更长的距离才能示教相关路径。

液压式冲压机同步的原理

- 1 记录相关液压式冲压机的移动情况。
- 2 在下一周期中激活待使用的记录。
- 3 用RAPID指令SyncToSensor激活相应的传感器同步。

4.1.11.2 系统参数的配置

简介

本节描述了如何配置相关参数才能在使用已记录的传感器曲线和一台液压式冲压機時取得最佳結果。首先要做的是用一般設定進行微調。如果系統未在使用DSQC377A編碼器，則參見第195頁的[无DSQC377A編碼器時的模擬輸入設定](#)；如果传感器正在使用編組輸入，則參見第195頁的[采用編組輸入的传感器設定](#)。第203頁的系統參數描述了相關的系統參數。

一般设定

该参数属于主题I / O下的配置类型现场总线命令。

参数	值
现场总线命令的类型为IIRFFP时，该实例的参数值。	10到15赫兹，更改该数值，从而在启动和停止时获得良好的准确度。

该参数属于主题运动下的配置类型路径传感器同步。

参数	值
同步类型	ROBOT_TO_HPRES

这些参数属于主题进程下的配置类型传感器系统。

参数	值
传感器启动信号	输入该I / O信号的名称
停止冲压機信号	输入该I / O信号的名称
同步警报信号	输入该I / O信号的名称

无DSQC377A編碼器時的模擬輸入設定

这些参数属于主题进程下的配置类型Can接口。

参数	值
虚拟传感器	是
位置信号	输入该模拟输入项的名称



注意

除位置信号外，其它所有信号都宜为空（即“”）。



提示

RAPID程序中不需要WaitSensor和DropSensor。

采用編組輸入的传感器設定

这些参数属于主题进程下的配置类型传感器系统。

参数	值
Pos位置編組IO尺度	定义每米的输入数据量；默认值被设置成10000。

下一页继续

4 Motion coordination

4.1.11.2 系统参数的配置

续前页

这些参数属于主题进程下的配置类型*Can*接口。

参数	值
虚拟传感器	是
位置信号	输入所用编组输入项的名称。



注意

除位置信号外，其它所有信号都宜为空（即“”）



提示

RAPID程序中不需要WaitSensor和DropSensor。

4.1.11.3 程序示例

概述

本节描述了为一台液压式冲压机编写程序时的典型编程周期。

程序示例

第一个冲压机周期

*sensor_start_signal*上的一道脉冲将启动记录数组中的保存位置。

在该周期中，相关机器人并未与冲压机同步。

```
ActUnit SSYNCl;  
WaitSensor SSYNCl;  
! Set up a recording for 2 seconds  
PrxStartRecord SSYNCl, 2, PRX_HPRESS_PROF;  
! Process waiting for sensor_start_signal  
! then waiting for press movement and record it during 2 sec.
```

第二个冲压机周期

需要*sensor_start_signal*上的一道脉冲来同步记录读数和每个周期的实际位置。

在打开冲压机期间，相关机器人会与冲压机同步移动。

```
PrxActivAndStoreRecord SSYNCl, 0, "profile.log";  
WaitSensor Ssync1;  
MoveL p10, v1000, z10, tool, \WObj:=wobj0;  
SyncToSensor Ssync1\On;  
MoveL p20, v1000, z20, tool, \WObj:=wobj0;  
MoveL p30, v1000, z20, tool, \WObj:=wobj0;  
SyncToSensor Ssync1\Off;
```

第三个冲压机周期

无需专用指令，但需要*sensor_start_signal*上的一道脉冲来同步记录读数和每个周期的实际位置。也可启动一次新的记录。

在打开冲压机期间，相关机器人会与冲压机同步移动。

```
WaitSensor Ssync1;  
MoveL p10, v1000, z10, tool, \WObj:=wobj0;  
SyncToSensor Ssync1\On;  
MoveL p20, v1000, z20, tool, \WObj:=wobj0;  
MoveL p30, v1000, z20, tool, \WObj:=wobj0;  
SyncToSensor Ssync1\Off;
```

4.1.12 用已记录的曲线来与注塑机同步

4.1.12.1 介绍

概述

本节描述了如何用已记录的机器曲线来改善机器人与一台注塑机之间的同步准确度。该曲线的作用是建立注塑路径模型。若不使用已记录的曲线，则机器人与机器模型之间需要有更长的距离才能示教相关路径。

注塑同步的原理

- 1 记录相关注塑机的移动情况。
- 2 在下一周期中激活待使用的记录。
- 3 用RAPID指令SynctoSensor激活相应的传感器同步。



提示

如果正在关闭注塑机，则可用监控来代替同步。更多信息请参见[第202页的监控](#)。

4.1.12.2 系统参数的配置

简介

本节描述了如何配置相关参数才能在使用已记录的传感器曲线和一台注塑机时取得最佳结果。首先要做的是用一般设定进行微调。如果系统未在使用DSQC377A编码器，则参见第199页的无DSQC377A编码器时的模拟输入设定；如果传感器正在使用编组输入，则参见第199页的采用编组输入的传感器设定。第203页的系统参数描述了相关的系统参数。

一般设定

该参数属于主题I / O下的配置类型现场总线命令。

参数	值
现场总线命令的类型为IIRFFP时，该实例的参数值。	10到15赫兹，更改该数值，从而在启动和停止时获得良好的准确度。

该参数属于主题运动下的配置类型路径传感器同步。

参数	值
同步类型	SYNC_TO_IMM

这些参数属于主题进程下的配置类型传感器系统。

参数	值
传感器启动信号	输入该I / O信号的名称
停止冲压机信号	输入该I / O信号的名称
同步警报信号	输入该I / O信号的名称

无DSQC377A编码器时的模拟输入设定

这些参数属于主题进程下的配置类型Can接口。

参数	值
虚拟传感器	是
位置信号	输入该模拟输入项的名称



注意

除位置信号外，其它所有信号都宜为空（即“”）。



提示

RAPID程序中不需要WaitSensor和DropSensor。

采用编组输入的传感器设定

这些参数属于主题进程下的配置类型传感器系统。

参数	值
Pos位置编组IO尺度	定义相关编组输入项的每米增量；默认值被设置成10000。

下一页继续

4 Motion coordination

4.1.12.2 系统参数的配置

续前页

这些参数属于主题进程下的配置类型*Can*接口。

参数	值
虚拟传感器	是
位置信号	输入所用编组输入项的名称。



注意

除位置信号外，其它所有信号都宜为空（即“”）



提示

RAPID程序中不需要WaitSensor和DropSensor。

4.1.12.3 程序示例

概述

本节描述了一台注塑机编写程序时的典型编程周期。

程序示例

第一个冲压周期

*sensor_start_signal*上的一道脉冲将启动记录数组中的保存位置。

在该周期中，相关机器人并未与冲压周期同步。

```
ActUnit SSYNCl;  
WaitSensor SSYNCl;  
! Set up a recording for 2 seconds  
PrxStartRecord SSYNCl, 2, PRX_PROFILE_T1;  
! Process waiting for sensor_start_signal  
! then waiting for press movement and record it during 2 sec.
```

第二个冲压周期

需要*sensor_start_signal*上的一道脉冲来同步记录读数和每个周期的实际位置。

在打开冲压周期期间，相关机器人会与冲压周期同步移动。

```
PrxActivAndStoreRecord SSYNCl, 0, "profile.log";  
WaitSensor Ssyncl;  
MoveL p10, v1000, z10, tool, \WObj:=wobj0;  
SyncToSensor Ssyncl\On;  
MoveL p20, v1000, z20, tool, \WObj:=wobj0;  
MoveL p30, v1000, z20, tool, \WObj:=wobj0;  
SyncToSensor Ssyncl\Off;
```

第三个冲压周期

无需专用指令，但需要*sensor_start_signal*上的一道脉冲来同步记录读数和每个周期的实际位置。也可启动一次新的记录。

在打开冲压周期期间，相关机器人会与冲压周期同步移动。

```
WaitSensor Ssyncl;  
MoveL p10, v1000, z10, tool, \WObj:=wobj0;  
SyncToSensor Ssyncl\On;  
MoveL p20, v1000, z20, tool, \WObj:=wobj0;  
MoveL p30, v1000, z20, tool, \WObj:=wobj0;  
SyncToSensor Ssyncl\Off;
```

4.1.13 监控

简介

当机器人离开注塑机或冲压机范围时，用户可采用监控来节省周期时间。在此情况下，当捡拾零件后的机器人离开注塑机范围时，该机器人便会启用“关闭注塑机”，而不是等到离开机器范围后才能启用“关闭注塑”。

通过设置系统参数同步警报信号所定义的输出信号，当其太靠近相关机器人时，监控下的系统便会停止相关注塑机。

SupSyncSensorOn的作用是监控带注塑机或冲压机的机器人的移动情况。通常会一直监控到该机器人离开注塑机或冲压机范围为止。有了监控后，用户便可关闭同步，然后在工件落入注塑机或收集到注塑机中时打开监控。SupSyncSensorOn可避免机器人和注塑机受损。

监控并不会停用相关同步。

示例

这种情况下您无法将相关传感器移动已定义位置，而是必须在您的RAPID程序中设置相应的外轴值。

```
p10.extax.eax_f:=sens10;
p20.extax.eax_f:=sens20;
p30.extax.eax_f:=sens30;
WaitSensor Ssync1;
MoveL p10, v1000, fine, tool, \WObj:=wobj0;
SupSyncSensorOn Ssync1, 150, -100, 650\SafetyDelay:=0;;
MoveL p20, v1000, z20, tool, \WObj:=wobj0;
MoveL p30, v1000, fine, tool, \WObj:=wobj0;
SupSyncSensorOff Ssync1;
```

Sens10是机器人位于p10时的预期机器位置（与机器人移动有关的机器移动模型），而sens20则是机器人位于p20时的预期机器位置。

系统将在传感器位置650到150之间进行监控，并在机器人与模具的间距小于100毫米时触发相应的输出。

Safetydist（此时为-100）是预期机器位置与真实机器位置之间的差值限值。该限值必须为负，即是说相关模型宜始终在真实机器前方移动。当机器位置逐渐减小时，该限值必须是最大负位置差值（以及最小先行距离）所对应的负值；当机器位置逐渐增大时，该限值必须是最小正位置差值（以及最小先行距离）所对应的正值。

4.1.14 系统参数

关于系统参数

本节从大体上描述了相关的系统参数。这些参数方面的更多信息请参见技术参考手册 - 系统参数。

现场总线命令

仅用于传感器同步：

这些是主题 *I / O* 下类型为现场总线命令的各种实例。

现场总线命令的类型	描述
每米计数量	外部装置运动的每米计数量。
同步分离	定义相关外部装置“从出现一个同步信号后”到“把一个新的同步信号接受为一个有效对象前”之间的最短移动距离。 若为传感器同步，则无需更改默认值。
队列跟踪距离	定义相应同步开关的放置位置（相对于传感器上0.0米处的放置位置）。 若为传感器同步，则无需更改默认值。
启动窗口宽度	定义启动窗口的大小。可用指令WaitSensor来关联该窗口内的对象。 若为传感器同步，则无需更改默认值。
IIRFFP	在左半平面上指定相关空洞真实部分的位置（以赫兹为单位）。

传感器系统

这些参数属于主题进程和类型传感器系统。

参数	描述
调节速度	在输入传感器同步时，用户必须将机器人速度调节成相应外部装置的速度。调节速度定义了相关机器人在首次运动中“赶上”该速度的速度（毫米 / 秒）。
最小距离	在系统自动丢弃一个关联对象前，该对象可能具有的最小距离（以毫米为单位）。 若为传感器同步，则无需更改默认值。 并非用于模拟同步。
最大距离	在系统自动丢弃一个关联对象前，该对象可能具有的最大距离（以毫米为单位）。 若为传感器同步，则无需更改默认值。 并非用于模拟同步。
传感器标称速度	相关外部装置的标称工作速度。如果该装置的速度超过了200毫米 / 秒，那么就必须增大该参数。
停止冲压信号	表明正在停止压制的相关数字输入信号的名称。用户需要该信号来安全停止机器人。
传感器启动信号	用于使已记录的曲线和新机器的移动相互同步的数字输入信号的名称。必须先设置该信号，然后才能开始移动机器。必须在冲压移动前100毫秒时触发该信号。
启动增减率	定义多少个计算步骤可能使位置错误超过最大先行距离。该增减期内的相关位置错误可能是最大先行距离的5倍。
同步警报信号	用于停止同步化机器的数字输出信号的名称。可在监控同步传感器时设置该信号。

下一页继续

4 Motion coordination

4.1.14 系统参数

续前页

CAN接口

这些参数属于主题进程和类型CAN接口。

参数	描述
已关联信号	用于连接的数字输入信号的名称。 并非用于模拟同步。
正信号	针对传感器位置的模拟输入信号的名称。
速度信号	针对传感器速度的模拟输入信号的名称。
零速度信号	表明相关传感器上速度为零的数字输入信号的名称。 并非用于模拟同步。
数据就绪信号	标明一批编码器单元的的数字输入信号的名称。 并非用于模拟同步。
Waitwobj信号	相关数字输出信号的名称，其表明需要与相关队列中的一个对象建立连接。 并非用于模拟同步。
Dropwobj信号	在编码器单元上丢下一个关联对象的数字输出信号的名称。 并非用于模拟同步。
PassStartW信号	相关数字输出信号的名称，其表明有一个对象在无关联的情况下越过了启动窗口。 并非用于模拟同步。
Pos更新时间	相关同步进程读取传感器位置时的时间（以毫秒为单位）。

运动规划器

这些参数属于主题运动和类型运动规划器。

参数	描述
路径分辨率	计算各个沿路径步骤的时段。
进程更新时间	相关传感器进程更新传感器位置上的机器人运动学情况的时间。
CPU载入均衡化	采用同步选项时需降低CPU载入均衡化。其默认值为2，但为了获得稳定的同步速度，采用同步选项时宜将其设置成1。

机械单元

这些参数属于主题运动和类型机械单元。

参数	描述
名称	该单元的名称（最多7个字符）。
在启动时激活	启动时会自动激活相应的传感器。
停用禁止	无法停用该传感器。

单一型

该参数属于主题运动和类型单一型。

参数	描述
机械结构	指定相关传感器的机械结构。

下一页继续

传输

该参数属于主题运动和类型传输。

参数	描述
旋转移动	指定相关传感器是旋转传感器（是）还是线性传感器（否）。

路径传感器同步

这些参数属于主题运动和类型路径传感器同步。其作用是设置相关外部装置的计算位置与实际位置之间的允许偏差，以及相关机器人的最小 / 最大TCP速度。

参数	描述
最大先行距离	相关外部装置的计算位置到实际位置之间所允许的最大先行距离。
最大滞后距离	相关外部装置的计算位置到实际位置之间所允许的最大滞后距离。
最大同步速度	允许的最大机器人TCP速度（米 / 秒）。
最小同步速度	允许的最小机器人TCP速度（米 / 秒）。

4 Motion coordination

4.1.15 I/O 信号

4.1.15 I/O 信号

概述

利用传感器同步功能提供的若干I / O信号，用户或RAPID程序可监测和控制编码器接口单元上的相应对象队列。有关方面为选项“传送器跟踪”设计了该对象队列，并使该队列具备了传感器同步之外的功能。既然我们把冲压机的每次关闭视为该对象队列中的一个对象，那么该对象队列的各种信号有时就能发挥出作用。

对象队列信号

下表展示了编码器单元DSQC 354（该单元会影响到该对象队列）中的各种I / O信号。

指令	描述
c1ObjectsInQ	编组输入展示了该对象队列中的对象数目。同步开关会登记这些对象，而系统也尚未丢弃这些对象。
c1Rem1PObj	会从该对象队列中移除第一个待定对象的数字输出项。待定对象是指位于队列之中、但未关联到某个工件上的对象。
c1RemAllPObj	会移除所有待定对象的数字输出。如果某个对象已被关联，那么就不会移除该对象。
c1DropWObj	数字输出将导致相应的编码器接口单元丢弃所跟踪的对象，并与其断开连接。系统会从队列中移除该对象。 请勿在RAPID代码中使用 <i>c1DropWObj</i> ，而是改用 <i>DropWobj</i> 指令。

4.1.16 RAPID组件

关于RAPID部件

此处概述了*Machine Synchronization*中的所有指令、函数和数据类型。
有关更多信息，请参阅 技术参考手册 - *RAPID*指令、函数和数据类型。

指令：

指令：	描述
DropSensor	将对象丢到传感器上
PrxActivAndStoreRecord	激活和保存已记录的曲线数据
PrxActivRecord	激活已记录的曲线数据
PrxDbgStoreRecord	保存和调试已记录的曲线数据
PrxDeactRecord	停用一项记录
PrxResetPos	重置相关传感器的零位
PrxResetRecords	重置并停用所有记录
PrxSetPosOffset	为相关传感器设置一个参考位置
PrxSetRecordSampleTime	设置记录一条曲线的样本时间
PrxSetSyncalarm	设置同步警报行为
PrxStartRecord	记录一条新曲线
PrxStopRecord	停止对曲线的记录
PrxStoreRecord	保存已记录的曲线数据
PrxUseFileRecord	使用已记录的曲线数据
SupSyncSensorOff	停止对同步化传感器的监控
SupSyncSensorOn	启动对同步化传感器的监控
SyncToSensor	与传感器同步
WaitSensor	等候传感器的连接

函数

函数	描述
PrxGetMaxRecordpos	获得最大传感器位置

数据类型

*Machine Synchronization*中不包括任何数据类型。

此页刻意留白

5 Motion Events

5.1 World Zones [608-1]

5.1.1 概述

目的

World Zones的作用是在机器人位于用户专门定义的区域时停止该机器人或设置一个输出信号。以下是一些应用示例：

- 当两台机器人的工作区域部分重叠时。可通过World Zones监控来安全地消除这两台机器人相撞的可能性。
- 当该机器人的工作区域内有某种永久性障碍或某些临时外部设备时。可创建一个禁区来防止机器人与此类设备相撞。
- 指明相关机器人正处在一个“允许用可编程逻辑控制器（PLC）来开始执行程序”的位置。

在程序执行期间和点动期间监控机器人移动时的一个全局区域。如果相关机器人的TCP触及该全局区域，或相关的轴触及了关节上的全局区域，那么就停止相关移动，并设置一个数字输出信号。



警告

出于安全考虑，用户不得使用本软件来保护人员——请使用硬件保护装备来保护人员。

其中包括

您可通过RobotWare选项World Zones来访问：

- 定义各种形状之体积的指令
- 在各轴坐标中定义关节区域的指令
- 定义和启用全局区域的指令

基本方法

这是设置World Zones的一般方式。第213页的代码示例用一个更详细的示例展示了其具体做法。

- 1 声明该全局区域是固定的还是临时的。
- 2 声明该形状变量。
- 3 定义该全局区域应具有的形状。
- 4 定义相应的全局区域（当达到相应体积时，系统就应停止相关机器人，或设置一个输出信号）。

限制

体积监控仅作用在TCP上，而机器人的任何其它部分都可能不知不觉地穿过这一体积。为了确保防止这种情况，您可对一个关节全局区域（由WZLimJointDef或WZHomeJointDef定义）进行监控。

下一页继续

5 Motion Events

5.1.1 概述

续前页

无法重新定义类型为wzstationary或wztemporary的变量。这些变量只能定义一次（用WZLimSup或WZDOSet进行定义）。

5.1.2 RAPID组件

数据类型

此处简述了World Zones中的每种数据类型。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各种数据类型。

数据类型	描述
wztemporary	wztemporary的作用是识别临时全局区域，并可用在RAPID程序中的任何位置。 可通过RAPID指令来禁用、重新启用或擦除临时全局区域。当载入一段新程序时，或当从MAIN例程的起点处开始执行程序时，系统便会自动擦除临时全局区域。
wzstationary	wzstationary的作用是识别固定全局区域，并仅能用在与事件“通电”相关联的一则事件例程中。定义事件例程方面的信息请参见操作员手册 - 带 <i>FlexPendant</i> 的 <i>IRC5</i> 。 固定全局区域会始终处于激活状态，而重启（先关闭电源然后再打开电源，或更改系统参数）则会再次激活此类区域。无法通过RAPID指令来禁用、启用或擦除固定全局区域。 如果涉及到安全问题，则应使用固定全局区域。
shapedata	shapedata的作用是描述一个全局区域的几何形状。 可将全局区域定义为4种不同的几何形状： <ul style="list-style-type: none"> • 一个方盒，所有侧面都与全局坐标系平行 • 一个圆柱体，与全局坐标系的z轴平行 • 一个球体 • 针对机器人轴和 / 或外轴的一个关节角区

指令：

此处简述了World Zones中的每条指令。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各条指令。

指令	描述
WZBoxDef	WZBoxDef的作用是定义这样一种体积：该体积呈方盒形，且此“盒子”的每一侧都与全局坐标系的各轴平行。其定义会保存在一个类型为shapedata的变量中。 也可将该体积定义为此“盒子”的相反部分（所有体积都位于此“盒子”之外）。
WZCylDef	WZCylDef的作用是定义这样一种体积：该体积呈圆柱形，且此“圆柱体”的轴与全局坐标系的z轴平行。其定义会保存在一个类型为shapedata的变量中。 也可将该体积定义为此“圆柱体”的相反部分（所有体积都位于此“圆柱体”之外）。
WZSphDef	WZSphDef的作用是定义一种球形体积。其定义会保存在一个类型为shapedata的变量中。 也可将该体积定义为此“球体”的相反部分（所有体积都位于此“球体”之外）。
WZLimJointDef	WZLimJointDef的作用是定义各轴的关节坐标，从而对相关工作区域进行限制。机器人轴和外轴都可设置坐标限值。 WZLimJointDef定义了每根轴的上限和下限。对旋转轴来说，这些限值的单位为“度”；对线性轴来说，这些限值的单位为“毫米”。 其定义保存在一个类型为shapedata的变量中。

下一页继续

5 Motion Events

5.1.2 RAPID组件

续前页

指令	描述
WZHomeJointDef	<p>WZHomeJointDef的作用是定义各轴的关节坐标，从而识别相关关节空间中的一个位置。机器人轴和外轴都可设置坐标限值。</p> <p>就每根轴而言，WZHomeJointDef定义了该区域中点的关节坐标以及距离该中点的区域Δ偏差。对旋转轴来说，这些坐标的单位为“度”；对线性轴来说，这些坐标的单位为“毫米”。</p> <p>其定义保存在一个类型为shapedata的变量中。</p>
WZLimSup	<p>WZLimSup的作用是定义和启用“在TCP抵达相应全局区域时停止相关机器人”，并届时附上一则错误消息。执行程序时和点动时都会激活这种监控。</p> <p>在调用WZLimSup时，您可指定其是一个保存在wzstationary变量中的固定全局区域，还是一个保存在wztemporary变量中的临时全局区域。</p>
WZDOSet	<p>WZDOSet的作用是定义和启用“在TCP抵达相应全局区域时设置一个数字输出信号”。</p> <p>在调用WZDOSet时，您可指定其是一个保存在wzstationary变量中的固定全局区域，还是一个保存在wztemporary变量中的临时全局区域。</p>
WZDisable	WZDisable的作用是禁用对某个临时全局区域的监控。
WZEnable	<p>WZEnable的作用是重新启用对某个临时全局区域的监控。</p> <p>当创建一个全局区域时，系统会自动启用该区域。只有在用WZDisable禁用此类区域的情况下，才有必要使用“启用”。</p>
WZFree	WZFree的作用是禁用和擦除对某个临时全局区域的监控。

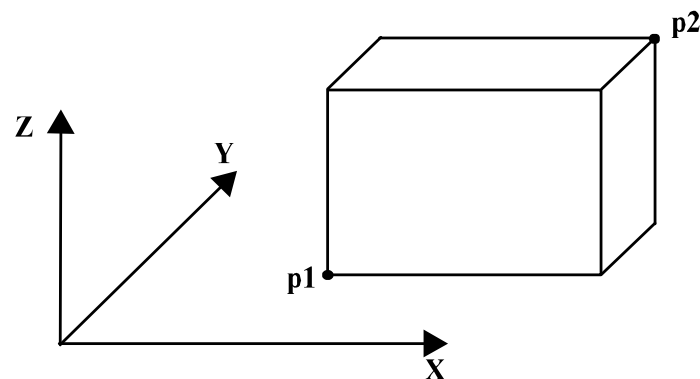
函数

World Zones不包括任何RAPID函数。

5.1.3 代码示例

创建受保护的盒子

为了避免相关的机器人TCP移入固定设备中，请围绕该设备设置一个固定全局区域。然后则宜把my_power_on例程与事件“通电”关联起来。至于具体的做法，则请阅读操作员手册 - 带 *FlexPendant* 的 *IRC5* 中的事件例程定义。



xx0300000178

```
VAR wzstationary obstacle;
PROC my_power_on()
  VAR shapedata volume;
  CONST pos p1 := [200, 100, 100];
  CONST pos p2 := [600, 400, 400];

  !Define a box between the corners p1 and p2
  WZBoxDef \Inside, volume, p1, p2;

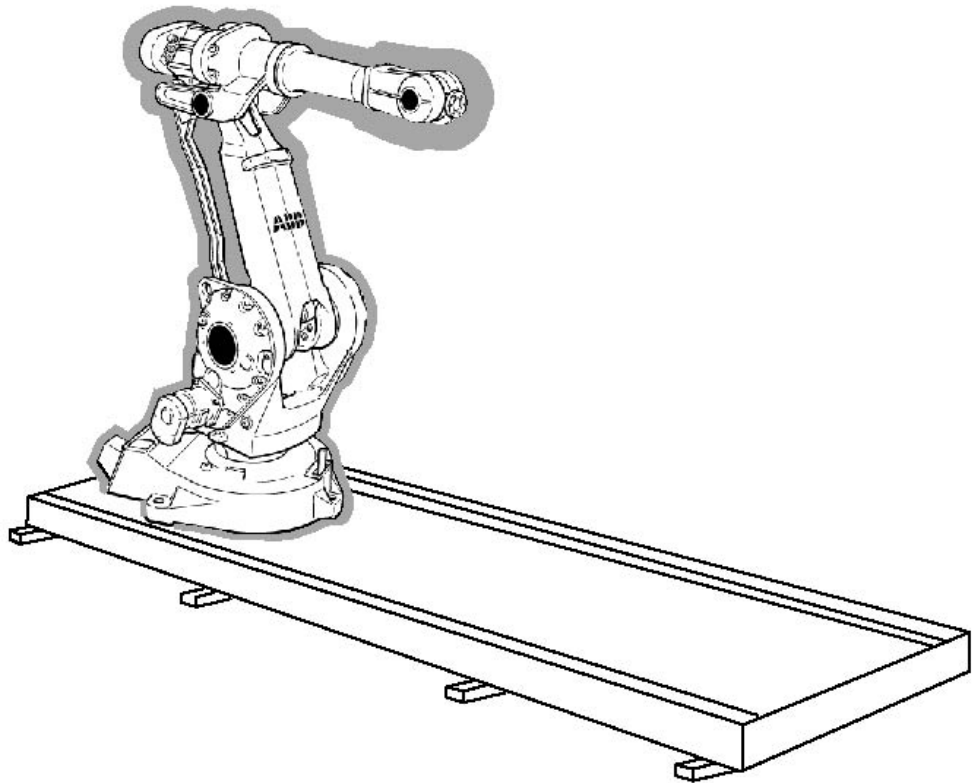
  !Define and enable supervision of the box
  WZLimSup \Stat, obstacle, volume;
ENDPROC
```

机器人就位时的信号

当两台机器人共享一个工作区域时，很重要的一点就是了解一台机器人何时会让出道路、使另一台机器人能自由移动。

此例定义了一个相关机器人处于安全位置时的起始位置，并设置了一个机器人处于其起始位置时的输出信号。相关机器人会立一条往返导轨上，而系统则会将其作为外轴1进行处理。其它外轴均未激活。

插图中的阴影区域展示了相应的全局区域。



xx0300000206

```
VAR wztemporary home;
PROC zone_output()
  VAR shapedata joint_space;

  !Define the home position
  CONST jointtarget home_pos := [[0, -20, 0, 0, 0, 0], [0, 9E9,
    9E9, 9E9, 9E9, 9E9]];

  !Define accepted deviation from the home position
  CONST jointtarget delta_pos := [[2, 2, 2, 2, 2, 2], [10, 9E9,
    9E9, 9E9, 9E9, 9E9]];

  !Define the shape of the world zone
  WZHomeJointDef \Inside, joint_space, home_pos, delta_pos;

  !Define the world zone, setting the
  !signal do_home to 1 when in zone
  WZDOSet \Temp, home \Inside, joint_space, do_home, 1;
ENDPROC
```

6 Motion functions

6.1 Independent Axes [610-1]

6.1.1 概述

目的

Independent Axes的作用，是在独立于本机器人系统中其它各轴的情况下移动一根轴。以下是一些应用示例：

- 移动夹持有一个对象的一根外轴（比如在机器人为该对象喷漆时旋转该对象）。
- 在外轴执行其它任务的同时执行一项机器人任务，从而节省周期时间。
- 连续旋转机器人轴6（用于抛光任务或类似任务）。
- 当某根轴朝同一方向旋转多圈后重置相应的测量系统。保存相对于物理回卷的周期时间。

如果某根轴被设置成独立模式，它便可独立移动。用户可以将一根轴先改为独立模式，然后又恢复成正常模式。

其中包括

您可通过RobotWare选项Independent Axes来访问：

- 设置独立模式和指定某轴移动情况的指令
- 改回正常模式和 / 或重置相关测量系统的一条指令
- 用于验证一根独立轴之状态的函数
- 用于配置的系统参数。

基本方法

这是独立移动一根轴的一般方式。第218页的代码示例用一个更详细的示例展示了其具体做法。

- 1 调用一条独立移动指令来把该轴设置成独立模式并移动该轴。
- 2 让相关机器人在独立轴移动时执行另一条指令。
- 3 当机器人和独立轴均已停止后，请将其中的独立轴重置为正常状态。

重置轴

即使不在独立模式下，轴也可能仅朝一个方向旋转，并最终丧失精确度。此时可用指令IndReset来重置相关测量系统。

建议当一根轴的电机朝同一方向旋转10000圈以上后，便重置该轴的测量系统。

限制

如果某个机械单元有一根轴正处于独立模式，则可能无法停用该机械单元。

无法点动独立模式下的各轴。

唯一能用作独立轴的机器人轴就是6号轴。在IRB 1600、2600和4600型号（ID版本除外）上，用户也可对轴4使用指令IndReset。

6 Motion functions

6.1.2 系统参数

6.1.2 系统参数

关于系统参数

此处简述了Independent Axes中的每个参数。更多信息请参见技术参考手册 - 系统参数中的各个参数。

机械臂

这些参数属于主题*Motion*下的类型*Arm*。

参数	描述
Independent Joint	决定了是否允许该轴使用独立模式的旗标。
Independent Upper Joint Bound	定义该关节在独立模式下运行时的工作区域上限。
Independent Lower Joint Bound	定义该关节在独立模式下运行时的工作区域下限。

传输

这些参数属于主题*Motion*下的类型*Transmission*。

参数	描述
Transmission Gear High	独立轴需要分辨率较高的传输齿轮比，因此该齿轮比被定义为 <i>Transmission Gear High</i> 除以 <i>Transmission Gear Low</i> 。如果 <i>Transmission Gear High</i> 被设置成相关机器人轴侧的嵌齿数目，而 <i>Transmission Gear Low</i> 被设置成电机侧的嵌齿数目，且无法使用更小的数字，那么相应的传输齿轮比就是正确的。
Transmission Gear Low	参见 <i>Transmission Gear High</i> 。

6.1.3 RAPID组件

数据类型

没有针对Independent Axes的数据类型。

指令：

此处简述了Independent Axes中的每条指令。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各条指令。

即使该轴此时正在移动，系统也会立即执行一条独立移动指令。如果在上一条独立移动指令尚未结束时就执行一条新的独立移动指令，那么新的指令会立即超驰旧的指令。

指令	描述
IndAMove	IndAMove（独立绝对位置移动）会把一根轴改为独立模式，然后将该轴移到指定位置。
IndCMove	IndCMove（独立连续移动）会把一根轴改为独立模式，然后开始按指定速度来连续移动该轴。
IndDMove	IndDMove（独立绝对 Δ 位置移动）会把一根轴改为独立模式，然后使该轴移动指定距离。
IndRMove	IndRMove（独立相对位置移动）会把一根轴改为独立模式，然后在一圈内将该轴移到一个特定位置。 由于忽略了相关位置的旋转信息，因此IndRMove永远不会旋转一轴圈以上。
IndReset	IndReset的作用是将一根独立轴改回正常模式。 IndReset能将旋转轴的测量系统转动许多轴圈。当逐渐远离逻辑位置0时，各位置的分辨率将有所下降，而回卷相应的轴则要耗费一定时间。通过移动相关测量系统，用户可在不对轴进行物理回卷的前提下维持相应的分辨率。 在调用IndReset，相关的独立轴和机器人都必须直立不动。

函数

此处简述了Independent Axes中的每则函数。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各则函数。

功能	描述
IndInpos	IndInpos指明了某根轴是否已抵达选定位置。
IndSpeed	IndSpeed指明了某根轴是否已达到选定速度。

6.1.4 代码示例

节省周期时间

A站中的对象有两处需要焊接。当相关机器人正在焊接另一个对象时，A站的外轴可将自身中的对象转动到合适位置，从而节省周期时间（而不是在移动外轴时让机器人白白等候）。

```
!Perform first welding in station A
!Call subroutine for welding
weld_stationA_1;

!Move the object in station A, axis 1, with
!independent movement to position 90 degrees
!at the speed 20 degrees/second
IndAMove Station_A,1\ToAbsNum:=90,20;

!Let the robot perform another task while waiting
!Call subroutine for welding
weld_stationB_1;

!Wait until the independent axis is in position
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;

!Perform second welding in station A
!Call subroutine for welding
weld_stationA_2;
```

用旋转轴6抛光

若要抛光一个对象，则可将机器人轴6设置成连续旋转。

将机器人轴6设置成独立模式，然后连续旋转该轴。将相关机器人移到您想抛光的区域上方，然后停止该机器人和该独立轴的移动，将模式改回正常模式。为了维持相应的分辨率，当该轴旋转了许多圈后请重置相关的测量系统。

请注意，若要使本例真正起到作用，就必须把rob1_6的参数*Independent Joint*设置成Yes。

```
PROC Polish()
!Change axis 6 of ROB_1 to independent mode and
!rotate it with 180 degrees/second
IndCMove ROB_1, 6, 180;

!Wait until axis 6 is up to speed
WaitUntil IndSpeed(ROB_1,6\InSpeed);
WaitTime 0.2;

!Move robot where you want to polish
MoveL p1,v10, z50, tool1;
MoveL p2,v10, fine, tool1;

!Stop axis 6 and wait until it's still
IndCMove ROB_1, 6, 0;
```

下一页继续

```
WaitUntil IndSpeed(ROB_1,6\ZeroSpeed);
WaitTime 0.2;

!Change axis 6 back to normal mode and
!reset measurement system (close to 0)
IndReset ROB_1, 6 \RefNum:=0 \Short;
ENDPROC
```

重置一根轴

此例展示了如何重置A站中轴1的测量系统。该测量系统会以整圈为单位来更改圈数，因此其靠近零位 ($\pm 180^\circ$)。

```
IndReset Station_A, 1 \RefNum:=0 \Short;
```

6.2 Path Recovery [611-1]

6.2.1 概述

目的

Path Recovery的作用是保存当前移动路径，执行某些机器人移动，然后恢复被中断的路径。一旦在路径移动期间发生错误或中断，这项功能就将发挥作用。可由一个错误处理器或一则中断例程来执行一项任务，然后重新创建相应的路径。

对电弧焊和胶合等应用而言，很重要的一点就是从相关机器人离开的位点处继续工作。如果相关机器人从起点处重新开始，那么就只能废弃相关工件了。

如果当机器人在工件内部时发生了一项进程错误，那么直接移动该机器人就可能导致碰撞。而采用了路径记录器后，该机器人便能沿着进来时的同一路径移出工件。

其中包括

您可通过RobotWare选项Path Recovery来访问：

- 在相应的错误或中断等级上暂停和继续协调同步移动模式的各条指令。
- 一件路径记录器，它能让TCP沿着移入时的同一路径移出。

限制

指令StorePath和RestoPath只会处理移动路径数据。必须保存相应的停止位置。

如果某次移动采用了路径记录器，那么就必须在软中断等级上执行此次移动，即是说必须在PathRecMoveBwd之前执行StorePath。

6.2.2 RAPID组件

数据类型

此处简述了Path Recovery中的每种数据类型。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各种数据类型。

数据类型	描述
pathrecid	pathrecid的作用是识别路径记录器的断点。

指令：

此处简述了Path Recovery中的每条指令。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各条指令。

指令	描述
StorePath	StorePath的作用是保存当发生错误或中断时正在执行的移动路径。 RobotWare基座包括了StorePath。
RestoPath	RestoPath的作用是恢复StorePath所保存的路径。 RobotWare基座包括了RestoPath。
PathRecStart	PathRecStart的作用是开始记录相关机器人的路径。路径记录器将保存执行相关机器人程序时的路径信息。
PathRecStop	PathRecStop的作用是停止记录相关机器人的路径。
PathRecMoveBwd	PathRecMoveBwd的作用是沿所记录的路径撤回相关机器人。
PathRecMoveFwd	PathRecMoveFwd的作用是将相关机器人撤回曾执行PathRecMoveBwd的位置。 也可提供一个后撤移动中被跳过的标识符，从而让相关机器人部分前移。
SyncMoveSuspend	SyncMoveSuspend的作用是暂停同步移动模式，并将系统设置成独立移动模式。
SyncMoveResume	SyncmoveResume的作用是从独立移动模式回到同步移动。

函数

此处简述了Path Recovery中的每则函数。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各则函数。

功能	描述
PathRecValidBwd	PathRecValidBwd的作用是检查相关路径记录器是否处于激活状态，以及是否有一条已记录的后撤路径可用。
PathRecValidFwd	PathRecValidFwd的作用是检查是否能用相关路径记录器向前移动。若能用路径记录器向前移动，则意味着之前肯定已经下令该路径记录器后撤。

6.2.3 保存当前路径

为何保存该路径？

使用Path Recovery的最简单方式就是仅保存“能在解决一项错误或类似行动后得以恢复”的当前路径。

假设电弧焊期间发生了一项错误。为了解决这一错误，用户可能必须从相关零件处移开相关机器人。在解决了这一错误后，又最好从机器人离开处继续焊接。而要做到这一点，就要在机器人离开相关路径前保存该机器人的路径信息和位置，然后便可在处理完错误后恢复这一路径并继续焊接。

基本方法

这是保存当前路径的一般方式：

- 1 在启动一个错误处理器或中断例程时：
 - 停止相关运动
 - 保存相关移动路径
 - 保存相应的停止位置
- 2 在启动一个错误处理器或中断例程时：
 - 移到已保存的停止位置处
 - 恢复相关移动路径
 - 开始相关移动

示例

此例展示了如何在处理错误时使用Path Recovery。首先保存相应的路径和位置，再校正相关错误，然后将相关机器人撤至适当位置，最后恢复相应路径。

```
MoveL p100, v100, z10, gun1;
...
ERROR
  IF ERRNO=MY_GUN_ERR THEN
    gun_cleaning();
  ENDIF
...
PROC gun_cleaning()
  VAR robtarget pl;

  !Stop the robot movement, if not already stopped.
  StopMove;

  !Store the movement path and current position
  StorePath;
  pl := CRobT(\Tool:=gun1\WObj:=wobj0);

  !Correct the error
  MoveL pclean, v100, fine, gun1;
  ...
  !Move the robot back to the stored position
  MoveL pl, v100, fine, gun1;
```

```

!Restore the path and start the movement
RestoPath;
StartMove;
RETRY;
ENDPROC

```

将路径保存到一套MultiMove系统中

在MultiMove系统中，相关机器人可在StorePath后用自变数KeepSync来保持同步移动模式。不过这些机器人无法从独立模式切换为同步模式，用户只能另辟蹊径。

在用自变数KeepSync设置一套Multimove系统后，该系统便能在StorePath等级上的同步模式、半协调模式和独立模式之间改动。请用指令SyncMoveResume和SyncMoveSuspend.来实现这种改动。

协调同步移动的“SyncArc”示例

此例展示了如何在一套MultiMove系统的错误处理器中使用Path Recovery和维持同步模式。为了使此例更加简单和泛用，我们使用了移动指令而非焊接指令。有一个定位器来旋转相关工作件。该SyncArc示例方面的更多信息请参见应用手册 - *MultiMove*。

T_ROB1任务程序

```

MODULE module1
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
PERS tasks all_tasks{3} := [{"T_ROB1"}, {"T_ROB2"}, {"T_STN1"}];
PERS wobjdata wobj_stn1 := [ FALSE, FALSE, "STN_1", [ [0, 0, 0],
[1, 0, 0, 0] ], [ [0, 0, 250], [1, 0, 0, 0] ] ];
TASK PERS tooldata tool1 := ...
CONST robtarg p100 := ...
CONST robtarg p199 := ...
PROC main()
...
SyncMove;
ENDPROC

PROC SyncMove()
MoveJ p100, v1000, z50, tool1;
WaitSyncTask sync1, all_tasks;
MoveL p101, v500, fine, tool1;
SyncMoveOn sync2, all_tasks;
MoveL p102\ID:=10, v300, z10, tool1 \WObj:=wobj_stn1;
MoveC p103, p104\ID:=20, v300, z10, tool1 \WObj:=wobj_stn1;
MoveL p105\ID:=30, v300, z10, tool1 \WObj:=wobj_stn1;
MoveC p106, p101\ID:=40, v300, fine, tool1 \WObj:=wobj_stn1;
SyncMoveOff sync3;
MoveL p199, v1000, fine, tool1;
ERROR
IF ERRNO = ERR_PATH_STOP THEN
gun_cleaning();
ENDIF

```

6 Motion functions

6.2.3 保存当前路径

续前页

```
UNDO
  SyncMoveUndo;
ENDPROC

PROC gun_cleaning()
  VAR robtarget p1;
  !Store the movement path and current position
  ! and keep synchronized mode.
  StorePath \KeepSync;
  p1 := CRobT(\Tool:=tool1 \WObj:=wobj_stn1);
  !Correct the error
  MoveL pclean1 \ID:=50, v100, fine, tool1 \WObj:=wobj_stn1;
  ...
  !Move the robot back to the stored position
  MoveL p1 \ID:=60, v100, fine, tool1 \WObj:=wobj_stn1;
  !Restore the path and start the movement
  RestoPath;
  StartMove;
  RETRY;
ENDPROC
ENDMODULE
```

T_ROB2任务程序

```
MODULE module2
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  PERS tasks all_tasks{3};
  PERS wobjdata wobj_stn1;
  TASK PERS tooldata tool2 := ...
  CONST robtarget p200 := ...
  CONST robtarget p299 := ...
  PROC main()
    ...
    SyncMove;
  ENDPROC
  PROC SyncMove()
    MoveJ p200, v1000, z50, tool2;
    WaitSyncTask sync1, all_tasks;
    MoveL p201, v500, fine, tool2;
    SyncMoveOn sync2, all_tasks;
    MoveL p202\ID:=10, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveC p203, p204\ID:=20, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveL p205\ID:=30, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveC p206, p201\ID:=40, v300, fine, tool2 \WObj:=wobj_stn1;
    SyncMoveOff sync3;
    MoveL p299, v1000, fine, tool2;
  ERROR
    IF ERRNO = ERR_PATH_STOP THEN
      gun_cleaning();
    ENDIF
  ENDPROC
ENDMODULE
```

下一页继续


```

UNDO
  SyncMoveUndo;
ENDPROC
PROC gun_cleaning()
  VAR robtarget p2;
  !Store the movement path and current position.
  StorePath \KeepSync;
  p2 := CRobT(\Tool:=tool2 \WObj:=wobj_stn1);
  !Correct the error
  MoveL pclean2 \ID:=50, v100, fine, tool2 \WObj:=wobj_stn1;
  ...
  !Move the robot back to the stored position.
  MoveL p2 \ID:=60, v100, fine, tool2 \WObj:=wobj_stn1;
  !Restore the path and start the movement
  RestoPath;
  StartMove;
  RETRY;
ENDPROC
ENDMODULE

```

T_STN1任务程序

```

MODULE module3
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  PERS tasks all_tasks{3};
  CONST jointtarget angle_neg20 :=[ [ 9E9, 9E9, 9E9, 9E9, 9E9,
    9E9], [ -20, 9E9, 9E9, 9E9, 9E9, 9E9] ];
  ...
  CONST jointtarget angle_340 :=[ [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9],[
    340, 9E9, 9E9, 9E9, 9E9, 9E9] ];
  PROC main()
    ...
    SyncMove;
    ...
  ENDPROC
  PROC SyncMove()
    MoveExtJ angle_neg20, vrot50, fine;
    WaitSyncTask sync1, all_tasks;
    ! Wait for the robots
    SyncMoveOn sync2, all_tasks;
    MoveExtJ angle_20\ID:=10, vrot100, z10;
    MoveExtJ angle_160\ID:=20, vrot100, z10;
    MoveExtJ angle_200\ID:=30, vrot100, z10;
    MoveExtJ angle_340\ID:=40, vrot100, fine;
    SyncMoveOff sync3;
  ERROR
    IF ERRNO = ERR_PATH_STOP THEN
      gun_cleaning();
    ENDIF
  ENDPROC
ENDMODULE

```

6 Motion functions

6.2.3 保存当前路径

续前页

```
UNDO
  SyncMoveUndo;
ENDPROC
PROC gun_cleaning()
  VAR jointtarget resume_angle;
  !Store the movement path and current angle.
  StorePath \KeepSync;
  resume_angle := CJointT();
  !Correct the error
  MoveExtJ clean_angle \ID:=50, vrot100, fine;
  ...
  !Move the robot back to the stored position.
  MoveExtJ resume_angle \ID:=60, vrot100, fine;
  !Restore the path and start the movement
  RestoPath;
  StartMove;
  RETRY;
ENDPROC
ENDMODULE
```

暂停和继续“SyncArc”示例中的同步移动

SyncMoveSuspend的作用是暂停同步移动模式，并将系统设置成独立或半协调移动模式。

SyncMoveResume的作用是再一次返回同步移动。

只有在执行了StorePath\KeepSync后才能使用这些指令。

T_ROB1

```
PROC gun_cleaning()
  VAR robtarget p1;
  !Store the movement path and current position
  ! and keep synchronized mode.
  StorePath \KeepSync;
  p1 := CRobT(\Tool:=tool1 \WObj:=wobj_stn1);
  !Move in synchronized motion mode
  MoveL p104 \ID:=50, v100, fine, tool1 \WObj:=wobj_stn1;
  SyncMoveSuspend;
  !Move in independent mode
  MoveL pclean1, v100, fine, tool1;
  ...
  !Move the robot back to the stored position
  SyncMoveResume;
  MoveL p1 \ID:=60, v100, fine, tool1 \WObj:=wobj_stn1;
  !Restore the path and start the movement
  RestoPath;
  StartMove;
  RETRY;
ENDPROC
```

T_ROB2

```
PROC gun_cleaning()
  VAR robtarget p2;
```

下一页继续

```

!Store the movement path and current position.
StorePath \KeepSync;
p2 := CRobT(\Tool:=tool2 \WObj:=wobj_stn1);
!Move in synchronized motion mode
MoveL p104 \ID:=50, v100, fine, tool2 \WObj:=wobj_stn1;
SyncMoveSuspend;
!Move in independent mode
MoveL pclean2 v100, fine, tool2;
...
!Move the robot back to the stored position.
SyncMoveResume;
!Move in synchronized motion mode
MoveL p2 \ID:=60, v100, fine, tool2 \WObj:=wobj_stn1;
!Restore the path and start the movement
RestoPath;
StartMove;
RETRY;
ENDPROC

```

T_STN1

```

PROC gun_cleaning()
VAR jointtarget resume_angle;
!Store the movement path and current angle.
StorePath \KeepSync;
resume_angle := CJointT();
!Move in synchronized motion mode
MoveExtJ p1clean_angle \ID:=50, vrot100, fine;
SyncMoveSuspend;
! Move in independent mode
MoveExtJ p2clean_angle,vrot, fine;
...
!Move the robot back to the stored position.
SyncMoveResume;
! Move in synchronized motion mode
MoveExtJ resume_angle \ID:=60, vrot100, fine;
!Restore the path and start the movement
RestoPath;
StartMove;
RETRY;
ENDPROC

```

6.2.4 路径记录

什么是路径记录器

路径记录器能储存大量移动指令，而利用这些指令，用户便能沿同一路径撤回相关机器人。

如何使用路径记录器

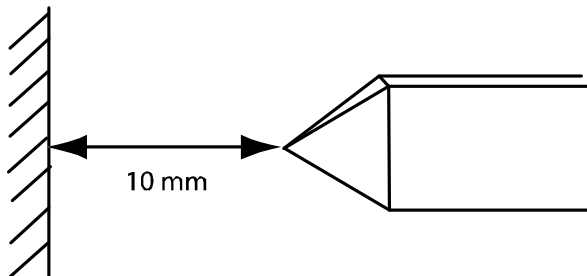
这是路径记录器的一般用法：

- 1 启动路径记录器
- 2 用常规移动、进程或指令来移动机器人
- 3 保存当前路径
- 4 沿已记录的路径后撤
- 5 解决错误
- 6 沿已记录的路径前进
- 7 恢复被中断的路径

抬高工具

当机器人撤回其自身导轨后，您可能希望避免工具擦刮到相关工件，比如想在电弧焊过程中避开焊缝等。

通过使用指令`PathRecMoveBwd`和`PathRecMoveFwd`中的自变数`ToolOffs`，您可为TCP设置一段偏移。该偏移会被设置在工具坐标上，这意味着如果将其设置成【0, 0, 10】，那么当相关工具沿已记录的路径后撤时，其将与相应工件相距10毫米。



xx0400000828



注意

当一套MultiMove系统处于同步模式时，如果正要抬高一件工具，那么所有任务都必须使用`ToolOffs`。

话说回来，如果您仅想抬高一件工具，那么就在其它任务中设置`ToolOffs=[0,0,0]`。

简单示例

如果p1与p4之间发生了一项错误，那么相关机器人就会返回p1来解决这一错误。当错误得到解决后，该机器人便会从发生错误处继续运行。

当没有任何错误就抵达p4时，系统便会关闭相应的路径记录器，然后相关机器人会在没有路径记录器的情况下从p4移到p5。

...

下一页继续

```

VAR pathrecid start_id;
...
MoveL p1, vmax, fine, tool1;
PathRecStart start_id;
MoveL p2, vmax, z50, tool1;
MoveL p3, vmax, z50, tool1;
MoveL p4, vmax, fine, tool1;
PathRecStop \Clear;
MoveL p5, vmax, fine, tool1;

ERROR
  StorePath;
  PathRecMoveBwd;
  ! Fix the problem
  PathRecMoveFwd;
  RestoPath;
  StartMove;
  RETRY;
ENDIF
...

```

复杂示例

本例中路径记录器有两种用途：

- 如果发生了一项错误，那么操作员可以选择退回p1或p2。一旦解决了相关错误，系统就会恢复被中断的移动过程。
- 如果未发生任何错误，那么系统就会用路径记录器将相关机器人从p4移到p1。当相关机器人处在一个难以移出的狭窄位置时，这种技巧会有所帮助。

请注意，如果在第一条移动指令期间（p1与p2之间）发生错误，那么就无法返回p2。如果操作员选择退回p2，那么就用PathRecValidBwd来查看这种做法是否可行。当相关机器人前进到其之前中断的位置时，则用PathRecValidFwd来查看这种做法是否可行（如果相关机器人始终不回退，那么就是它已经就位了）。

```

...
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;
MoveL p4, vmax, fine, tool1;

! Use path record to move safely to p1
StorePath;
PathRecMoveBwd \ID:=origin_id
  \ToolOffs:=[0,0,10];
RestoPath;
PathRecStop \Clear;

```

下一页继续

6.2.4 路径记录

续前页

```
Clear Path;
Start Move;

ERROR
StorePath;

! Ask operator how far to back up
TPReadFK choice,"Extract to:", stEmpty, stEmpty,
    stEmpty, "Origin", "Corner";

IF choice=4 THEN
    ! Back up to p1
    PathRecMoveBwd \ID:=origin_id
        \ToolOffs:=[0,0,10];
ELSEIF choice=5 THEN
    ! Verify that it is possible to back to p2,
    IF PathRecValidBwd(\ID:=corner_id) THEN
        ! Back up to p2
        PathRecMoveBwd \ID:=corner_id
            \ToolOffs:=[0,0,10];
    ENDIF
ENDIF

! Fix the problem

! Verify that there is a path record forward
IF PathRecValidFwd() THEN
    ! Return to where the path was interrupted
    PathRecMoveFwd \ToolOffs:=[0,0,10];
ENDIF

! Restore the path and resume movement
RestoPath;
StartMove;
RETRY;
...
```

恢复路径记录器

如果停止了该路径记录器，那么就能在不丢失其历史记录的情况下从同一位置重新启动该记录器。

下例中的PathRecMoveBwd指令使相关机器人退回到了p1。在重启路径记录器时，如果相关机器人正处在p2之外的其它位置，那么它就无法退回到p1。

更多信息请参见技术参考手册 - RAPID指令、函数和数据类型中关于PathRecStop的章节。

```
...
MoveL p1, vmax, z50, tool1;
PathRecStart id1;
MoveL p2, vmax, z50, tool1;
PathRecStop;
MoveL p3, vmax, z50, tool1;
```

下一页继续

```

MoveL p4, vmax, z50, tool1;
MoveL p2, vmax, z50, tool1;
PathRecStart id2;
MoveL p5, vmax, z50, tool1;
StorePath;
PathRecMoveBwd \ID:=id1;
RestoPath;
...

```

协调同步移动的“SyncArc”示例

此例展示了如何在MultiMove系统的错误处理过程中使用路径记录器。

此例中有两台机器人在对同一工件进行电弧焊。为了使此例更加简单和泛用，我们使用了移动指令而非焊接指令。有一个定位器来旋转该工件。

该SyncArc示例方面的更多信息请参见应用手册 - *MultiMove*。

T_ROB1任务程序

```

MODULE module1
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  PERS tasks all_tasks{3} := [{"T_ROB1"}, {"T_ROB2"}, {"T_STN1"}];
  PERS wobjdata wobj_stn1 := [ FALSE, FALSE, "STN_1", [ [0, 0, 0],
    [1, 0, 0, 0] ], [ [0, 0, 250], [1, 0, 0, 0] ] ];
  TASK PERS tooldata tool1 := ...
  CONST robtarget pl00 := ...
  CONST robtarget pl99 := ...
  PROC main()
    ...
    SyncMove;
  ENDPROC

  PROC SyncMove()
    WaitSyncTask sync1, all_tasks;
    MoveJ pl00, v1000, z50, tool1;
    ! Start recording
    PathRecStart HomeROB1;
    MoveL pl01, v500, fine, tool1;
    SyncMoveOn sync2, all_tasks;
    MoveL pl02\ID:=10, v300, z10, tool1 \WObj:=wobj_stn1;
    MoveC pl03, pl04\ID:=20, v300, z10, tool1 \WObj:=wobj_stn1;
    MoveL pl05\ID:=30, v300, z10, tool1 \WObj:=wobj_stn1;
    MoveC pl06, pl01\ID:=40, v300, fine, tool1 \WObj:=wobj_stn1;
    !Stop recording
    PathRecStop \Clear;
    SyncMoveOff sync3;
    MoveL pl99, v1000, fine, tool1;
  ERROR
    ! Weld error in this program task
    IF ERRNO = AW_WELD_ERR THEN
      gun_cleaning();
    ENDIF
  ENDPROC

```

6.2.4 路径记录

续前页

```
ENDIF
UNDO
  SyncMoveUndo;
ENDPROC
PROC gun_cleaning()
  VAR robtarget pl;
  !Store the movement path
  IF IsSyncMoveOn() THEN
    StorePath \KeepSync;
  ELSE
    StorePath;
  ENDIF
  !Move this robot backward to p100.
  PathRecMoveBwd \ID:=HomeROB1 \ToolOffs:=[0,0,10];
  !Correct the error
  MoveJ pclean1 ,v100, fine, tool1;
  ...
  !Move the robot back to p100
  MoveJ p100, v100, fine, tool1;
  PathRecMoveFwd \ToolOffs:=[0,0,10];
  !Restore the path and start the movement
  RestoPath;
  StartMove;
  RETRY;
ENDPROC
ENDMODULE
```

T_ROB2任务程序

```
MODULE module2
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  PERS tasks all_tasks{3};
  PERS wobjdata wobj_stn1;
  TASK PERS tooldata tool2 := ...
  CONST robtarget p200 := ...
  CONST robtarget p299 := ...
  PROC main()
    ...
    SyncMove;
  ENDPROC
  PROC SyncMove()
    WaitSyncTask sync1, all_tasks;
    MoveJ p200, v1000, z50, tool2;
    PathRecStart HomeROB2;
    MoveL p201, v500, fine, tool2;
    SyncMoveOn sync2, all_tasks;
    MoveL p202\ID:=10, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveC p203, p204\ID:=20, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveL p205\ID:=30, v300, z10, tool2 \WObj:=wobj_stn1;
    MoveC p206, p201\ID:=40, v300, fine, tool2 \WObj:=wobj_stn1;
```

下一页继续


```

    PathRecStop \Clear;
    SyncMoveOff sync3;
    MoveL p299, v1000, fine, tool2;
ERROR
    IF ERRNO = ERR_PATH_STOP THEN
        gun_move_out();
    ENDIF
UNDO
    SyncMoveUndo;
ENDPROC
PROC gun_move_out()
    IF IsSyncMoveOn() THEN
        StorePath \KeepSync;
    ELSE
        StorePath;
    ENDIF
    ! Move this robot backward to p201
    PathRecMoveBwd \ToolOffs:=[0,0,10];
    ! Wait for the other gun to get clean
    PathRecMoveFwd \ToolOffs:=[0,0,10];
    !Restore the path and start the movement
    RestoPath;
    StartMove;
    RETRY;
ENDPROC
ENDMODULE

```

T_STN1任务程序

```

MODULE module3
    VAR syncident sync1;
    VAR syncident sync2;
    VAR syncident sync3;
    PERS tasks all_tasks{3};
    CONST jointtarget angle_neg20 :=[ [ 9E9, 9E9, 9E9, 9E9, 9E9,
        9E9], [ -20, 9E9, 9E9, 9E9, 9E9, 9E9] ];
    ...
    CONST jointtarget angle_340 :=[ [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9],[
        340, 9E9, 9E9, 9E9, 9E9, 9E9] ];
    PROC main()
        ...
        SyncMove;
        ...
    ENDPROC
    PROC SyncMove()
        WaitSyncTask sync1, all_tasks;
        MoveExtJ angle_neg20, vrot50, fine;
        PathRecStart HomeSTN1;
        SyncMoveOn sync2, all_tasks;
        MoveExtJ angle_20\ID:=10, vrot100, z10;
        MoveExtJ angle_160\ID:=20, vrot100, z10;
        MoveExtJ angle_200\ID:=30, vrot100, z10;
    ENDPROC

```

下一页继续

6.2.4 路径记录

续前页

```
MoveExtJ angle_340\ID:=40, vrot100, fine;
PathRecStop \Clear;
SyncMoveOff sync3;
ERROR
  IF ERRNO = ERR_PATH_STOP THEN
    gun_move_out();
  ENDIF
UNDO
  SyncMoveUndo;
ENDPROC
PROC gun_move_out()
  !Store the movement
  IF IsSyncMoveOn() THEN
    StorePath \KeepSync;
  ELSE
    StorePath;
  ENDIF
  !Move the manipulator backward to angle_neg 20
  PathRecMoveBwd \ToolOffs:=[0,0,0];
  ...
  !Wait for the gun to get clean
  PathRecMoveFwd \ToolOffs:=[0,0,0];
  RestoPath;
  StartMove;
  RETRY;
ENDPROC
```

6.3 Path Offset [612-1]

6.3.1 概述

目的

Path Offset的作用是根据来自传感器的输入项来在线调节机器人路径。若提供有Path Offset指令组，那么就能用这些传感器输入项来对比和调节相应的机器人路径。

其中包括

您可通过RobotWare选项Path Offset来访问：

- 数据类型corrdescr
- 指令CorrCon、CorrDiscon、CorrClear和CorrWrite
- 函数CorrRead

基本方法

这是设置Path Offset的一般方式。第238页的代码示例例举了其具体做法。

- 1 声明相应的校正发生器。
- 2 关联相应的校正发生器。
- 3 定义一则决定了相关偏移量的软中断例程（该例程会被写入相应的校正发生器中）。
- 4 定义一项用来频繁调用该软中断例程的中断。
- 5 用该校正项调用一条移动指令。系统会反复校正相关路径。



注意

如果用\Corr开关相互颠倒地调用了两条或更多条移动指令，那么很重要的一点就是了解相关机器人每次从一个精确点启动时是否重置了所有\Corr偏移量。所以在使用精确点时，相关控制器在第二条移动指令期间并不清楚相关路径是否已有偏移。为避免行为失常，我们建议\Corr开关仅与区域一同使用，而不要使用精确点。

限制

有可能同时关联上若干个校正发生器（比如一个沿Z轴校正，另一个沿Y轴校正），但最多只能同时关联5个校正发生器。

若重启了控制器，则必须重新定义相应的校正发生器。当控制器重启后，系统不会保留之前的定义和关联。

这些指令仅能用于运动任务。

6 Motion functions

6.3.2 RAPID组件

6.3.2 RAPID组件

数据类型

此处简述了Path Offset中的每种数据类型。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各种数据类型。

数据类型	描述
corrdescr	corrdescr是一个校正发生器描述符，其用途是作为相应校正发生器的引用项。

指令：

此处简述了Path Offset中的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
CorrCon	CorrCon会激活路径校正。调用CorrCon将会关联上一个校正发生器。一旦关联成功，系统就能用新的偏移输入项（比如来自传感器的偏移输入项）来不断校正相关路径。
CorrDiscon	CorrDiscon会停用路径校正。调用CorrDiscon将会断开一个校正发生器。
CorrClear	CorrClear会停用路径校正。调用CorrClear将会断开所有校正发生器。
CorrWrite	CorrWrite会设置相应的路径校正。调用CorrWrite将会为一个校正发生器设置相应的偏移值。

函数

此处简述了Path Offset中的每则函数。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各则函数。

功能	描述
CorrRead	CorrRead会读取一个校正发生器所作的总校正。

6.3.3 相关的RAPID功能

自变数\Corr

可为部分移动指令设置\Corr。当执行这些移动指令时，系统便会启用路径校正。

下列指令拥有可选自变数\Corr：

- MoveL
- MoveC
- SearchL
- SearchC
- TriggL（仅当相关控制器配备了基本功能Fixed Position Events时）
- TriggC（仅当相关控制器配备了基本功能Fixed Position Events时）
- CapL（仅当相关控制器配备了选项Continuous Application Platform时）
- CapC（仅当相关控制器配备了选项Continuous Application Platform时）
- ArcL（仅当相关控制器配备了选项RobotWare Arc时）
- ArcC（仅当相关控制器配备了选项RobotWare Arc时）

至于这些指令方面的更多信息，则请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

中断

若要用Path Offset来创建程序，您就得有能力处理各种中断。中断方面的更多信息请参见技术参考手册 - *RAPID*语言概览。

6.3.4 代码示例

带校正的直线移动

此例展示了如何编写一条带在线路径校正的线性路径。这里的做法是每秒中断5次，并调用一则软中断例程来校正偏移量。

程序代码

```
VAR intnum int_nol;
VAR corrdescr id;
VAR pos sens_val;
PROC PathRoutine()
  !Connect to the correction generator
  CorrCon id;

  !Setup a 5 Hz timer interrupt.
  CONNECT int_nol WITH UpdateCorr;
  ITimer\Single, 0.2, int_nol

  !Position for start of contour tracking
  MoveJ p10,v100,z10,tool1;

  !Run MoveL with correction.
  MoveL p20,v100,z10,tool1\Corr;

  !Remove the correction generator.
  CorrDiscon id;

  !Remove the timer interrupt.
  IDelete int_nol;
ENDPROC
TRAP UpdateCorr
  !Call a routine that read the sensor
  ReadSensor sens_val.x, sens_val.y, sens_val.z;

  !Execute correction
  CorrWrite id, sens_val;

  !Setup interrupt again
  IDelete int_nol;
  CONNECT int_nol WITH UpdateCorr;
  ITimer\Single, 0.2, int_nol;
ENDTRAP
```

7 Motion Supervision

7.1 Collision Detection [613-1]

7.1.1 概述

目的

“碰撞检测”是一种软件选项，可以减少相关机器人承受的碰撞力度。这有助于避免机器人和外部设备受到严重损伤。



警告

“碰撞检测”无法在全速碰撞下保护设备。

描述

软件选项“碰撞检测”会对相关机器人实施基于型号的高敏感度监控，从而对碰撞进行识别。根据在相关机器人上刻意施加的力的状况，您既可对这种敏感度进行微调，也可以开启和关闭这种敏感度。由于相关机器人上的力会在程序执行过程中波动，因此您可在程序代码中将该敏感度设置成“在线”。

碰撞检测比普通的监控更为敏感，同时还有一些额外的特性。当检测到一次碰撞时，相关机器人会立即停止，并沿其路径反向移动一小段距离来释放余力。当接受了一则碰撞错误消息后，系统便能继续执行相应的移动，而无需按下相关控制器上的“电机开启”。

其中包括

您可通过RobotWare选项“碰撞检测”来访问：

- 这些系统参数的作用是定义是否宜激活“碰撞检测”，以及宜采用何种“碰撞检测”敏感度（若没有该选项，您就只能在自动模式下打开和关闭检测）
- 针对敏感度在线变化的指令：`MotionSup`

基本方法

机器人移动时默认始终激活“碰撞检测”，这意味着许多情况下您无需采取任何主动措施就能使用“碰撞检测”。

必要时您可打开和关闭“碰撞检测”，或通过两种途径更改其敏感度：

- 可通过RAPID指令`MotionSup`进行在线临时更改
- 通过系统参数进行永久性更改。

7 Motion Supervision

7.1.2 限制

7.1.2 限制

负载定义

为了恰当地检测出碰撞，用户必须正确定义相关机器人的有效负载。



提示

使用“负载识别”来定义有效负载。更多信息请参见操作员手册 - 带 *FlexPendant* 的 *IRC5*。

仅用于机器人轴

只有机器人轴才能使用“碰撞检测”，导轨运动、轨道站或其它任何外轴均无法使用这一功能。

独立关节

当至少有一根轴在独立关节模式下运行时，系统便会停用碰撞检测（即使是作为独立关节而运行的外轴也不例外）。

软伺服器

如果在软伺服器模式下使用相关机器人，那么就可能在无碰撞的情况下触发碰撞检测，因此我们建议在这种模式下使用机器人时关闭碰撞检测。

机器人移动前不作改动

如果使用RAPID指令`MotionSup`来关闭碰撞检测，那么只有当相关机器人开始移动时，这一操作才会真正生效，因此在相关机器人移动之前，数字输出项`MotSupOn`在程序开始时的数值可能会出乎意料。

反向移动距离

相关机器人在碰撞后的反向移动距离与碰撞前的运动速度成正比。如果反复发生低速碰撞，那么相关机器人就可能倒退足够远的距离来释放碰撞应力，所以在未触发监控的情况下可能无法点动机器人。此时要临时关闭“碰撞检测”，然后点动该机器人以远离障碍。

反向移动前的延时

如果在程序执行期间发生刚性碰撞，那么相关机器人可能要等待几秒才能反向移动。

导轨上的机器人

当把相关机器人安装在一条导轨上时，如果该导轨正在移动，那么就宜停用碰撞检测；如果不停用，那么即使当时没有碰撞，也可能在导轨移动时触发碰撞检测。

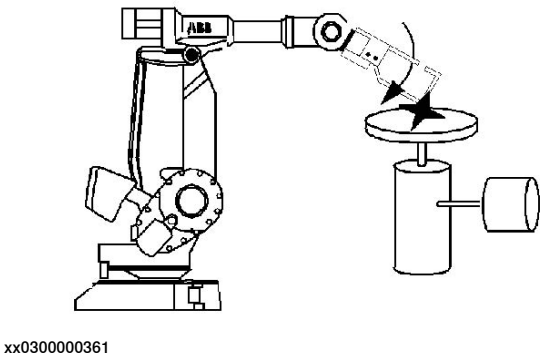
7.1.3 碰撞时的情况

概述

当触发碰撞检测时，相关机器人会尽快停止，然后通过反向移动来消除余力。相关程序会停止，并随之出现一则错误消息。相关机器人仍会停留在状态电机开启，因此待接受了该碰撞错误消息后，便能继续执行相关程序。

下图展示了一次典型的碰撞。

碰撞图



一次碰撞后的机器人行为

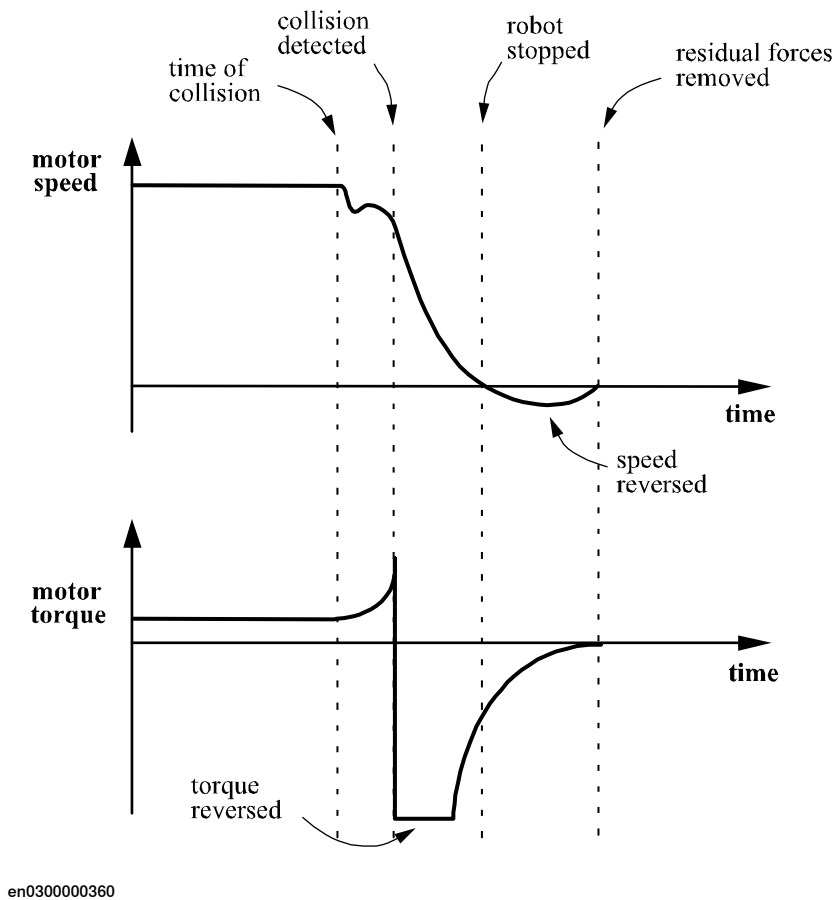
该表展示了一次碰撞后的事件顺序。其序列说明请详见下图。

当.....	那么.....
检测到碰撞	为了停止相关机器人，系统会反向旋转相应电机，并采用机械制动
相关机器人已停止	为了消除碰撞或卡滞时可能存在的任何余力，相关机器人会沿相关路径反向一小段距离
消除了余力	相关机器人会再次停止，但仍然处于电机开启状态

7 Motion Supervision

7.1.3 碰撞时的情况
续前页

速度与扭矩图



7.1.4 附加信息

运动错误处理

碰撞错误处理方面的更多信息请参见技术参考手册 - *RAPID*语言内核。

7 Motion Supervision

7.1.5.1 系统参数

7.1.5 配置和编写设施

7.1.5.1 系统参数

关于系统参数

这些碰撞检测参数无需重启便可生效。

这些参数方面的更多信息请参见技术参考手册 - 系统参数。

Motion Supervision

这些参数属于主题*Motion*下的类型*Motion Supervision*。

参数	描述
Path Collision Detection	按执行程序时的需要来打开或关闭碰撞检测。 <i>Path Collision Detection</i> 被默认设置成On。
Jog Collision Detection	按点动时的需要来打开或关闭碰撞检测。 <i>Jog Collision Detection</i> 被默认设置成On。
Path Collision Detection Level	按指定百分数修改执行程序时所需的碰撞检测监控等级。较大的百分数意味着该函数的敏感度较低。 <i>Path Collision Detection Level</i> 被默认设置成100%。
Jog Collision Detection Level	按指定百分数修改点动时所需的碰撞检测监控等级。较大的百分数意味着该函数的敏感度较低。 <i>Jog Collision Detection Level</i> 被默认设置成100%。
Collision Detection Memory	定义碰撞后的机器人要沿相关路径反向移动多远（以秒为单位）。与低速机器人相比，碰撞前移动较快的机器人会移动得更远。 <i>Collision Detection Memory</i> 被默认设置成75毫秒。
Manipulator Supervision	按IRB 340和IRB 360的需要来打开或关闭松臂检测监控。 <i>Manipulator Supervision</i> 被默认设置成On。
Manipulator Supervision Level	按机械臂IRB 340和IRB 360的需要来修改松臂检测的监控等级。较大的数值意味着该函数的敏感度较低。 <i>Manipulator Supervision Level</i> 的数值被默认设置成100%。

Motion Planner

这些参数属于主题*Motion*下的类型*Motion Planner*。

参数	描述
Motion Supervision Max Level	将最大等级设置成总的可更改碰撞检测微调等级。其默认设置成300%。

General RAPID

这些参数属于主题*Controller*下的类型*General RAPID*。

参数	描述
Collision Error Handler	启用针对碰撞的RAPID错误处理。 <i>Collision Error Handler</i> 被默认设置成Off。 碰撞错误处理方面的更多信息请参见技术参考手册 - RAPID语言内核。

7.1.5.2 RAPID组件

指令：

此处简述了“碰撞检测”中的各项指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
MotionSup	MotionSup的作用是： <ul style="list-style-type: none">• 激活或停用“碰撞检测”。只有当路径碰撞检测被设置成“开启”时，用户才能进行这种操作。• 按指定百分数（1%到300%）修改监控等级。较大的百分数意味着该函数的敏感度较低。

7 Motion Supervision

7.1.5.3 Signals

7.1.5.3 Signals

数字输出

此处简述了“碰撞检测”中的数字输出项。更多信息请参见技术参考手册 - 系统参数中的各个数字输出项。

数字输出	描述
MotSupOn	当“碰撞检测”处于激活状态时， <i>MotSupOn</i> 会偏高；当“碰撞检测”处于未激活状态时，则会偏低。 请注意，状态变化会在开始运动时生效，因此如果激活了“碰撞检测”且相关机器人正在移动，那么 <i>MotSupOn</i> 就会偏高。如果停止相关机器人并关闭“碰撞检测”，那么 <i>MotSupOn</i> 仍会偏高；而当该机器人开始移动时， <i>MotSupOn</i> 则会切换为低值。
MotSupTrigg	触发碰撞检测时 <i>MotSupTrigg</i> 会偏高，并将高值一直保持到用FlexPendant示教器接受了相应的错误代码为止。

7.1.6 何时使用Collision Detection

7.1.6.1 设置系统参数

激活监控

为了能在程序执行期间使用“碰撞检测”，用户必须将参数路径碰撞检测设置成开启。
为了能在点动期间使用“碰撞检测”，用户必须将参数点动碰撞检测设置成开启。

定义监控等级

将参数路径碰撞检测等级设置成你想要的百分数，以作为程序执行期间的默认值。
将参数点动碰撞检测等级设置成你想要的百分数，以作为点动期间的默认值。

7.1.6.2 FlexPendant示教器的调节监控

调节速度监控等级

“碰撞检测”采用了一种可变监控等级，且低速时比高速时更为敏感。出于这一缘故，用户在正常操作期间宜无需微调这一函数，不过可以通过打开和关闭该函数的方式来微调监控等级。

点动和程序执行期间可单独微调各个参数。[第244页的系统参数](#)对这些参数做了描述。

在FlexPendant示教器上设置点动监控

在FlexPendant示教器上的ABB菜单中选择控制面板，然后轻击监控。

不论是编程路径还是点动，用户均既能打开或关闭监控，又能调节相应的敏感度。按百分数来设置敏感度等级，较大的数值意味着该函数的敏感度较低。

如果在对话框中关闭了点动运动监控并执行了一段程序，那么执行该程序时“碰撞检测”仍会处于激活状态。



注意

这些监控设定对应着类型为*Motion Supervision*的系统参数。正如上文所述，用户可用FlexPendant示教器上的监控设定来设置它们，此外也可用RobotStudio或FlexPendant示教器的配置编辑器或“快速设置机械单元”菜单来更改它们。

7.1.6.3 用RAPID程序调节监控

默认值

如果用相关系统参数来激活“碰撞检测”，那么程序执行期间的“碰撞检测”将默认处于激活状态（微调值100%）。系统会自动设置这些数值：

- 当使用重启模式重置系统时。
- 当载入一段新程序时。
- 当从起点开始执行程序时。



注意

如果在相关系统参数和RAPID指令中设置了微调值，那么两者的数值都要纳入考虑。
示例：如果在相关系统参数中将该微调值设置成150%，在相关RAPID指令中将该微调值设置成200%，那么由此得出的微调等级就是300%。

临时停用监控

如果执行程序时有外力会影响到相关机器人，那么就用以下指令临时停用相关监控：

```
MotionSup \Off;
```

重新激活监控

如果已临时停用了相关监控，那么可用以下指令来激活监控：

```
MotionSup \On;
```



注意

如果已用系统参数停用了相关监控，那么就无法用RAPID指令来激活监控。

微调

执行程序时可用指令*MotionSup*来微调监控等级。此处是按基本微调的百分数来设置微调值，即是说100%对应着基本值。更高的百分数意味着系统敏感度更低。

此例中的一条指令将相关监控等级增加到了200%。

```
MotionSup \On \TuneValue:=200;
```

7 Motion Supervision

7.1.6.4 如何避免误触发

7.1.6.4 如何避免误触发

关于误触发

由于相关监控被设置得十分敏感，因此如果负载数据有误，或相关机器人会承受较大的加工力，那么就可能会触发监控。

待采取的行动

如果.....	那么.....
有效负载的定义有误	使用“负载识别”来定义它。更多信息请参见操作员手册 - 带 <i>FlexPendant</i> 的 <i>IRC5</i> 。
有效负载具有较大的质量或惯量	提高监控等级
因臂负载（电缆之类的物品）而导致触发	手动定义臂负载，或提高监控等级
相关应用涉及到许多外部加工力	按30%的幅度来逐步提高点动时和执行程序时的监控等级，直至您不再收到相应的错误代码为止。
外部加工力都只是暂时的	用指令 <code>MotionSup</code> 来提高相应的监控等级，或临时关闭相应的函数。
其它全部失效	关闭“碰撞检测”。

8 Communication

8.1 FTP Client [614-1]

8.1.1 FTP Client介绍

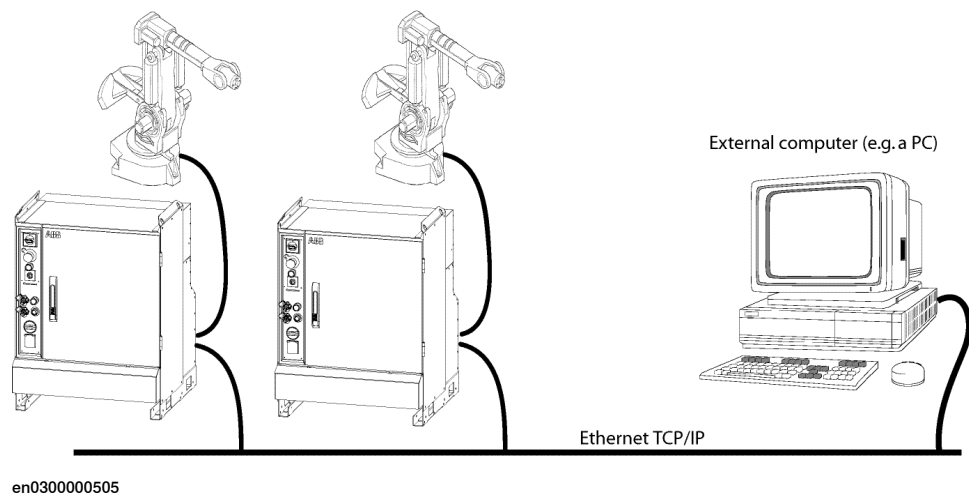
目的

FTP Client的作用是启用相关机器人来访问远程挂载磁盘，比如一台PC上的某个硬盘驱动器。

此处是一些应用示例：

- 备份到一台远程计算机中。
- 从一台远程计算机载入程序。

网络图



描述

多台机器人可通过网络来访问同一台计算机。

一旦配置了FTP应用层协议，用户就能像访问控制器的内部硬盘那样来访问相应的远程计算机。

其中包括

您可通过RobotWare选项FTP Client来访问系统参数类型*Application protocol*及其参数：*Name*、*Type*、*Transmission protocol*、*Server address*、*Trusted*、*Local path*、*Server path*、*Username*和*Password*

基本方法

这是FTP Client的一般用法。第254页的示例用一个更详细的示例展示了其具体用法。

- 1 配置一项应用层协议，以便指明相关机器人可在远程计算机上访问的某个硬盘或目录。
- 2 像读取和写入控制器内部硬盘那样来读取和写入相应的远程计算机。

下一页继续

8 Communication

8.1.1 FTP Client介绍

续前页

要求

该外部计算机必须具有：

- TCP/IP堆栈
- FTP服务器

限制

当使用FTP Client时，一个文件名称的最大长度为99个字符。

当使用FTP Client时，包括文件名称在内的一条文件路径最多长达200个字符。整条路径——而不仅仅是服务器路径——都要包括在这200个字符中。当命令备份到挂载磁盘上时，相关最大路径必须包括此次备份中创建的所有目录。

示例

参数	值
Local path	pc:
Server path	C:\robot_1

- 在pc:/Backups/Backup_20130109保存一份备份
(27个字符)
- 该PC上的路径将是C:\robot_1\Backups\Backup_20130109
(34个字符)
- 该备份中的最长文件路径为
C:\robot_1\Backups\Backup_20130109\RAPID\TASK1\PROGMOD\myprogram.mod
(54 + 13个字符)

此例中的最大路径长度乍眼一看似乎是27个字符，但实际上是67个字符。

8.1.2 系统参数

应用层协议

此处简述了用于配置一项应用层协议的相关参数。更多信息请参见[第253页的系统参数](#)中的各个参数。

这些参数属于主题*Communication*下的类型*Application protocol*。

参数	描述
Name	相关应用层协议的名称。
Type	应用层协议的类型。 将其设置成“FTP”。
Transmission protocol	宜采用的传输协议名称。 对FTP而言，这始终为“TCP/IP”。
Server address	带有FTP服务器的计算机的IP地址。
Trusted	该旗标决定了是否宜信任这台计算机，如果失去连接，那么是否就宜停止相关程序。
Local path	定义将在机器人上调用哪个共享单元。相关参数值必须以一个冒号 (:) 结尾。 举例来说，如果该单元被命名为“pc:”，那么test.prg在该单元上的名称就将是pc:test.prg
Server path	相关远程计算机上有待连接的磁盘或文件夹的名称。 如果没有指定，那么相关应用层协议就将引用与FTP服务器共享的目录。 注意：如果与类型为Distinct FTP、FileZilla或MS IIS的一台FTP服务器进行通信，那么就不宜指定输出路径。
Username	机器人登录相关远程计算机时所用的用户名。 必须在该FTP服务器上设置用户账户。
Password	机器人登录相关远程计算机时所用的密码。 请注意，所有能访问相关系统参数的人均可看到此处输入的密码。

传输协议

已配置了一项名为“TCP/IP”的传输协议，但用户不能更改该协议。相应的FTP应用层协议会使用这项传输协议。

8 Communication

8.1.3 示例

8.1.3 示例

配置示例

此例展示了如何为FTP配置一项应用层协议。

参数	值
Name	我的FTP协议
Type	FTP
Transmission protocol	TCPIP1
Server address	100.100.100.100
Trusted	否
Local path	pc :
Server path	C:\robot_1
Username	机器人1
Password	机器人1

注：如果与类型为Distinct FTP、FileZilla或MS IIS的一台FTP服务器进行通信，那么就宜从*Server path*的数值中排除输出路径。

FlexPendant示教器的示例

此例展示了如何用FlexPendant示教器来备份到相应的远程PC上。我们假设按上文所示的配置示例来进行配置。

- 1 轻击**ABB**，然后选择备份并恢复。
- 2 轻击备份当前系统。
- 3 将备份保存到pc:/Backup/Backup_20031008（该PC上的路径是C:\robot_1\Backup\Backup_20031008）。

RAPID代码示例

此例展示了如何用控制器上的一段RAPID程序来打开相应远程PC上的文件C:\robot_1\files\file1.doc。我们假设按上文所示的配置示例来进行配置。

```
Open "HOME:" \File:= "pc:/files/file1.doc", file;
```

8.2 NFS Client [614-1]

8.2.1 NFS Client介绍

目的

NFS Client的作用是启用相关机器人来访问远程挂载磁盘，比如一台PC上的某个硬盘驱动器。

此处是一些应用示例：

- 备份到一台远程计算机中。
- 从一台远程计算机载入程序。

描述

多台机器人可通过网络来访问同一台计算机。

一旦配置了NFS应用层协议，用户就能像访问控制器的内部硬盘那样来访问相应的远程计算机。

其中包括

您可通过RobotWare选项NFS Client来访问系统参数类型应用层协议及其各个参数：名称、Type、传输协议、服务器地址、受信任、本地路径、服务器路径、用户ID和编组ID。

基本方法

这是NFS Client的一般用法。[第254页的示例](#)用一个更详细的示例展示了其具体用法。

- 1 配置一项应用层协议，以便指明相关机器人可在远程计算机上访问的某个硬盘或目录。
- 2 像读取和写入控制器内部硬盘那样来读取和写入相应的远程计算机。

操作前提

该外部计算机必须具有：

- TCP/IP堆栈
- NFS服务器

限制

当使用NFS Client时，一个文件名称的最大长度为99个字符。

当使用NFS Client时，包括文件名称在内的一条文件路径也最多长达99个字符。整条路径——而不仅仅是服务器路径——都要包括在这99个字符中。当命令备份到挂载磁盘上时，相关最大路径必须包括此次备份中创建的所有目录。

示例

参数	值
Local path	pc:
Server path	C:\robot_1

- 在pc:/Backups/Backup_20130109保存一份备份
(27个字符)

下一页继续

8.2.1 NFS Client介绍

续前页

- 该PC上的路径将是C:\robot_1\Backups\Backup_20130109
(34个字符)
- 该备份中的最长文件路径为
C:\robot_1\Backups\Backup_20130109\RAPID\TASK1\PROGMOD\myprogram.mod
(54 + 13个字符)

此例中的最大路径长度乍眼一看似乎是27个字符，单实际上是67个字符。

8.2.2 系统参数

应用层协议

此处简述了用于配置一项应用层协议的相关参数。更多信息请参见第257页的系统参数中的各个参数。

这些参数属于主题通信下的类型应用层协议。

参数	描述
名称	相关应用层协议的名称。
类型	应用层协议的类型。 将其设置成“NFS”。
传输协议	宜采用的传输协议名称。 对NFS而言，这始终为“TCP/IP”。
服务器地址	带有NFS服务器的计算机的IP地址。
受信任	该旗标决定了是否宜信任这台计算机，如果失去连接，那么是否就宜停止相关程序。
本地路径	定义将在机器人上调用哪个共享单元。相关参数值必须以一个冒号(:) 结尾。 举例来说，如果该单元被命名为“pc:”，那么test.prg在该单元上的名称就将是pc:test.prg
服务器路径	从相关远程计算机上输出的磁盘或文件夹的名称。 必须为NFS指定服务器路径。
用户ID	被NFS协议作为授权用户访问某特定服务器的一种方式。 如果未使用该参数（PC上就通常不使用该参数），那么就将其设置成默认值0。 请注意，一个控制器上的所有配件都必须采用相同的用户ID。
编组ID	被NFS协议作为授权用户访问某特定服务器的一种方式。 如果未使用该参数（PC上就通常不使用该参数），那么就将其设置成默认值0。 请注意，一个控制器上的所有配件都必须采用相同的编组ID。

传输协议

已配置了一项名为“TCP/IP”的传输协议，但用户不能更改该协议。相应的NFS应用层协议会使用这项传输协议。

8.2.3 示例

配置示例

此例展示了如何为NFS配置一项应用层协议。

参数	值
名称	我的NFS协议
类型	NFS
传输协议	TCP/IP
服务器地址	100.100.100.100
受信任	否
本地路径	pc :
服务器路径	C:\robot_1
用户ID	机器人1
编组ID	机器人1

FlexPendant示教器的示例

此例展示了如何用FlexPendant示教器来备份到相应的远程PC上。我们假设按上文所示的配置示例来进行配置。

- 1 轻击**ABB**，然后选择备份并恢复。
- 2 轻击备份当前系统。
- 3 将备份保存到pc:/Backup/Backup_20031008（该PC上的路径是C:\robot_1\Backup\Backup_20031008）。

RAPID代码示例

此例展示了如何用控制器上的一段RAPID程序来打开相应远程PC上的文件C:\robot_1\files\file1.doc。我们假设按上文所示的配置示例来进行配置。

```
Open "HOME:" \File:= "pc:/files/file1.doc", file;
```

8.3 PC Interface [616-1]

8.3.1 PC Interface介绍

目的

PC Interface的作用是实现控制器与一台PC之间的通信。

当通过带RobotStudio的LAN来连接一个控制器时，用户需要使用选项PC Interface。

有了PC Interface，用户就能从一台PC上发送和接收数据，从而实现以下等方面的用途：

- 备份。
- 生产统计记录。
- 呈现在一台PC上的操作员信息。
- 从一个PC操作员接口向相关机器人发送命令。
- 在相关控制器上实现操作的RobotStudio插件。



注意

如果通过服务端口进行连接，那么则可在没有选项PC Interface的情况下使用此项功能。

其中包括

您可通过RobotWare选项PC Interface来访问：

- 一个以太网通信接口，某些ABB软件产品会使用这一接口。

基本方法

使用PC Interface的一般方式就好比在一台PC上设置一项PC SDK客户端应用。更多信息请参见<http://developercenter.robotstudio.com>。

8.3.2 发送RAPID的变量

SCWrite指令

可通过指令SCWrite (*Superior Computer Write*) 来向一台PC上的客户端应用发送永久变量。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型。

该PC上必须有一项能订阅相关信息（即相关机器人所接收或发送的信息）的客户端应用。

代码示例

在此例中，相关机器人把各对象移到了某个位置处，而受控于相关PC的某种工艺则能在该位置上处理这些对象。当对象处理完毕后，相关机器人便会将其移到下一站处。当相关对象就位时以及已将该对象移到下一站时，相关程序会用SCWrite来通知相应的PC，并向该PC发送一则消息，以便说明已处理了多少对象。

发送方所用的RAPID模块

```
VAR rmqslot destination_slot;
VAR user_def

RMQFindSlot destination_slot, "RMQ_Task2";

WHILE TRUE DO
    ! Wait for next object
    WaitDI di1,1;

    ! Call first routine
    move_obj_to_pos();

    ! Send message to PC that object is in position
    user_def = 0;
    in_position:=TRUE;
    RMQSendMessage destination_slot, in_position \UserDef:=user_def;

    ! Wait for object to be ready
    WaitDI di2,1;

    ! Call second routine
    move_obj_to_next();

    ! Send message to PC that object is gone
    in_position:=FALSE;
    RMQSendMessage destination_slot, in_position \UserDef:=user_def;

    ! Inform PC how many object has been handled
    nbr_objects:= nbr_objects+1;
    user_def = 1;
    RMQSendMessage destination_slot, nbr_objects \UserDef:=user_def;

ENDWHILE
```

下一页继续

接收方所用的PC SDK

```
public void ReceiveObjectPosition()
{
    const string destination_slot = "RMQ_Task2";
    IpcQueue queue = Controller.Ipc.CreateQueue(destination_slot,
        16, Ipc.MaxMessageSize);

    // Until application is closed
    while (uiclose)
    {
        IpcMessage message = new IpcMessage();
        IpcReturnType retValue = IpcReturnType.Timeout;

        retValue = queue.Receive(1000, message);
        if (IpcReturnType.OK == retValue)
        {
            string receivemessage = message.Data.ToString().ToLower();
            // if message.UserDef is 0 means Object position data else
            // number of objects
            if (message.UserDef == 0)
            {
                if (receivemessage == "true")
                {
                    // Object is in position
                }
                else
                {
                    // Object is not in position
                }
            }
            else
            {
                // number of objects in receivemessage
            }
        }
    }
}
```

8.3.3 使用PC接口的ABB软件

概述

PC接口为相关控制器与连接以太网的PC提供了一个相互间的通信接口。

ABB公司的各种软件应用都或可使用这项功能。请注意，上述产品只是在例举使用PC接口的应用，实际上还不止这些产品。

RobotStudio

RobotStudio是一款随机器人一同交付的软件产品。当通过LAN端口进行连接时，其某些功能需要PC Interface才能使用。

下表例举了某些当您拥有PC Interface时才能使用的RobotStudio功能：

功能	描述
事件记录器	可在相关PC上显示或记录错误消息和类似事件。
RAPID编辑器	允许进行不遵从相关PC控制器的在线编辑。

有关更多信息，请参阅 操作员手册 - *RobotStudio*。

8.4 Socket Messaging [616-1]

8.4.1 Socket Messaging介绍

目的

Socket Messaging的作用是允许RAPID程序员通过TCP/IP网络协议在各台计算机之间传输应用数据。一个套接字代表了一条独立于当前所用网络协议的通用通信通道。“套接字通信”是源于Berkeley所发布软件Unix的一套标准，而除Unix外，Microsoft Windows等平台也支持该项标准。有了Socket Messaging，机器人控制器上的RAPID程序就能与另一台计算机上的C/C++程序等进行通信。

其中包括

您可通过RobotWare选项Socket Messaging来访问在各计算机之间的套接字通信所需的RAPID数据类型、指令和函数。

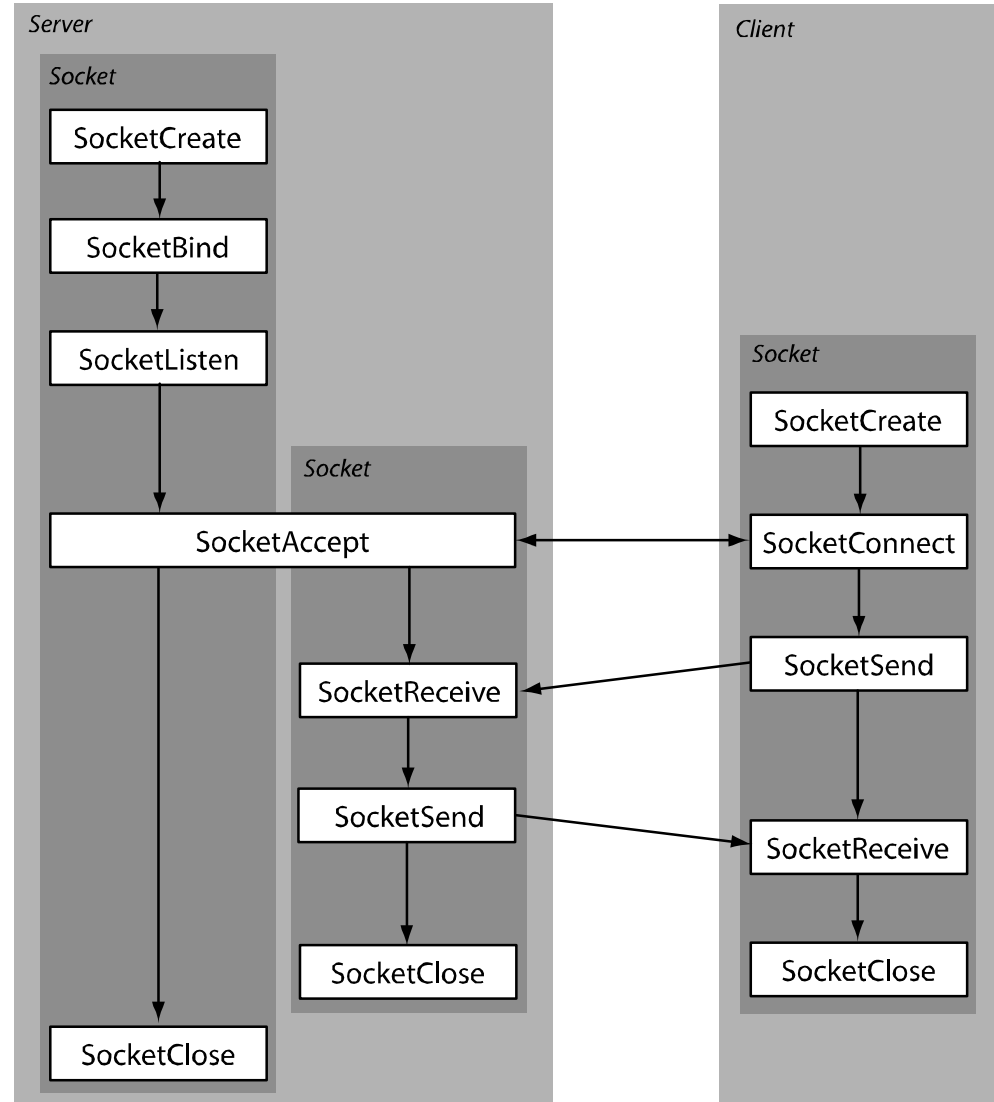
基本方法

这是Socket Messaging的一般用法。[第268页的代码示例](#)用一个更详细的示例展示了其具体用法。

- 1 在客户端和服务端上分别创建一个套接字。机器人控制器可以是客户端，也可以是服务器。
- 2 在相关服务器上使用SocketBind和SocketListen，使其对连接请求作好准备。
- 3 命令相关服务器接受外来的套接字连接请求。
- 4 从相关客户端提出套接字连接请求。
- 5 在客户端与服务器之间发送和接收数据。

8.4.2 套接字通信的示意图

套接字通信图



en0600003224



提示

若无必要，则请勿创建和关闭套接字。通信完毕前请一直开启相应的套接字。出于TCP/IP功能之故，在SocketClose后，该套接字要过一段时间后会真正关闭。

8.4.3 关于套接字消息发送的技术实情

概述

在用RAPID功能“套接字消息发送”与一个非RAPID任务的客户端或服务器进行通信时，掌握某些执行方式可能会为用户带来帮助。

无字符串终止

当发送一则数据消息时，系统会在该消息中发送一个无字符串终止符。所发送的字节数等同于编程语言C中函数`strlen(str)`的返回之后。

消息的意外合并

如果发送了两则消息，且两次发送之间没有任何延时，那么第二则消息就可能附加到第一则消息上，从而形成一则大消息而非两则消息。为避免这种情况，如果相关客户端 / 服务器正在接收消息，那么使用来自数据接收方的接受消息。

不可打印字符

如果某个并非RAPID任务的客户端需要从某RAPID任务的一段字符串中接收不可打印字符（二进制数据），那么便可用RAPID按下例所示进行接收。

```
SocketSend socket1 \Str:="\0D\0A";
```

更多信息请参见技术参考手册 - *RAPID*语言内核的节字符串文字。

8.4.4 RAPID组件

数据类型

此处简述了套接字消息发送的每种数据类型。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各种数据类型。

数据类型	描述
套接字装置	一件套接字装置，用于与网络上的其它计算机进行通信。
套接字状态	可包含一个socketdev变量的状态信息。

用于客户端的指令

此处简述了套接字消息发送客户端所用的每条指令。更多信息请参见技术参考手册 - *RAPID* 指令、函数和数据类型中的各条指令。

指令	描述
套接字创建	创建一个新的套接字，并赋予其一个socketdev变量。
套接字连接	向一台远程计算机提出连接请求。客户端将用其连接相应的服务器。
套接字发送	通过套接字连接而向某台远程计算机发送数据。这些数据既可以是string或rawbytes变量，也可以是byte数组。
套接字接收	接收数据，并将其保存在一个string或rawbytes变量中，或保存在一个byte数组中。
套接字关闭	关闭一个套接字，随之释放所有资源。



提示

不要在SocketSend之后直接使用SocketClose。先等候接受信息，然后再关闭套接字。

用于服务器的指令

除SocketConnect外，套接字消息发送服务器与客户端都使用着同一套指令。此外服务器还会使用下列指令：

指令	描述
套接字绑定	将套接字与相关服务器上的一个指定端口号绑定起来。该服务器会用来定义“用（该服务器上的）哪个端口监听某一连接”。该IP地址定义了一台物理计算机，而该端口则定义了通往该计算机上某一程序的一条逻辑通道。
套接字监听	使该计算机作为一台服务器，并接受外来的连接。其将监听SocketBind所指定端口上的某一连接。
套接字接受	接受一项外来连接请求。服务器将用其来接受相关客户端的请求。



注意

必须先启动服务器应用，然后才启动客户端应用，这样才能在任一客户端执行SocketConnect前先执行指令SocketAccept。

函数

此处简述了套接字消息发送的每则函数。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各则函数。

功能	描述
套接字获取状态	返回在套接字上执行的最后一条指令（已创建、已连接、边界、监听和关闭）。 <i>SocketGetStatus</i> 不会检测来自外部RAPID的改动（比如断开的连接）。

8.4.5 代码示例

客户端 / 服务器通信的示例

此例展示了客户端和服务端彼此通信时所需的程序代码。

将在FlexPendant示教器上写入该服务器：

```
Client wrote - Hello server
Client wrote - Shutdown connection
```

将在FlexPendant示教器上写入该客户端：

```
Server wrote - Message acknowledged
Server wrote - Shutdown acknowledged
```

此例中的客户端和服务端都使用了RAPID程序，而在实际中，人们往往会在一台PC（或类似的计算机）上运行其中一段程序，并将该程序写为另一种程序语言。

客户端的代码示例，用IP地址192.168.0.2联系服务器：

```
! WaitTime to delay start of client.
! Server application should start first.
WaitTime 5;
VAR socketdev socket1;
VAR string received_string;
PROC main()
    SocketCreate socket1;
    SocketConnect socket1, "192.168.0.2", 1025;
    ! Communication
    SocketSend socket1 \Str:="Hello server";
    SocketReceive socket1 \Str:=received_string;
    TPWrite "Server wrote - " + received_string;
    received_string := "";
    ! Continue sending and receiving
    ...
    ! Shutdown the connection
    SocketSend socket1 \Str:="Shutdown connection";
    SocketReceive socket1 \Str:=received_string;
    TPWrite "Server wrote - " + received_string;
    SocketClose socket1;
ENDPROC
```

服务器的代码示例（带IP地址192.168.0.2）：

```
VAR socketdev temp_socket;
VAR socketdev client_socket;
VAR string received_string;
VAR bool keep_listening := TRUE;
PROC main()
    SocketCreate temp_socket;
    SocketBind temp_socket, "192.168.0.2", 1025;
    SocketListen temp_socket;
    WHILE keep_listening DO
        ! Waiting for a connection request
        SocketAccept temp_socket, client_socket;
        ! Communication
        SocketReceive client_socket \Str:=received_string;
```

下一页继续

```

        TPWrite "Client wrote - " + received_string;
        received_string := "";
        SocketSend client_socket \Str:="Message acknowledged";
        ! Shutdown the connection
        SocketReceive client_socket \Str:=received_string;
        TPWrite "Client wrote - " + received_string;
        SocketSend client_socket \Str:="Shutdown acknowledged";
        SocketClose client_socket;
    ENDWHILE
    SocketClose temp_socket;
ENDPROC

```

错误处理器示例

下列错误处理器将对上电失败或断开连接进行处理。

针对上例所用客户端的错误处理器：

```

! Error handler to make it possible to handle power fail
ERROR
    IF ERRNO=ERR_SOCKET_TIMEOUT THEN
        RETRY;
    ELSEIF ERRNO=ERR_SOCKET_CLOSED THEN
        SocketClose socket1;
        ! WaitTime to delay start of client.
        ! Server application should start first.
        WaitTime 10;
        SocketCreate socket1;
        SocketConnect socket1, "192.168.0.2", 1025;
        RETRY;
    ELSE
        TPWrite "ERRNO = "\Num:=ERRNO;
        Stop;
    ENDIF

```

针对上例所用服务器的错误处理器：

```

! Error handler for power fail and connection lost
ERROR
    IF ERRNO=ERR_SOCKET_TIMEOUT THEN
        RETRY;
    ELSEIF ERRNO=ERR_SOCKET_CLOSED THEN
        SocketClose temp_socket;
        SocketClose client_socket;
        SocketCreate temp_socket;
        SocketBind temp_socket, "192.168.0.2", 1025;
        SocketListen temp_socket;
        SocketAccept temp_socket, client_socket;
        RETRY;
    ELSE
        TPWrite "ERRNO = "\Num:=ERRNO;
        Stop;
    ENDIF

```

8.5 RAPID Message Queue 【包括在616-1、623-1中】

8.5.1 RAPID Message Queue介绍

目的

RAPID Message Queue的作用是与另一项RAPID任务或使用PC SDK的PC应用进行通信。

此处是一些应用示例：

- 两项RAPID tasks之间的发送数据。
- 一项RAPID task与一项PC应用之间的发送数据。

可针对中断模式或同步模式来定义RAPID Message Queue。默认设置为中断模式。

其中包括

RobotWare选项包括了RAPID Message Queue功能：

- PC Interface
- Multitasking

您可通过RAPID Message Queue来访问用于发送和接收数据的RAPID指令、函数和数据类型。

基本方法

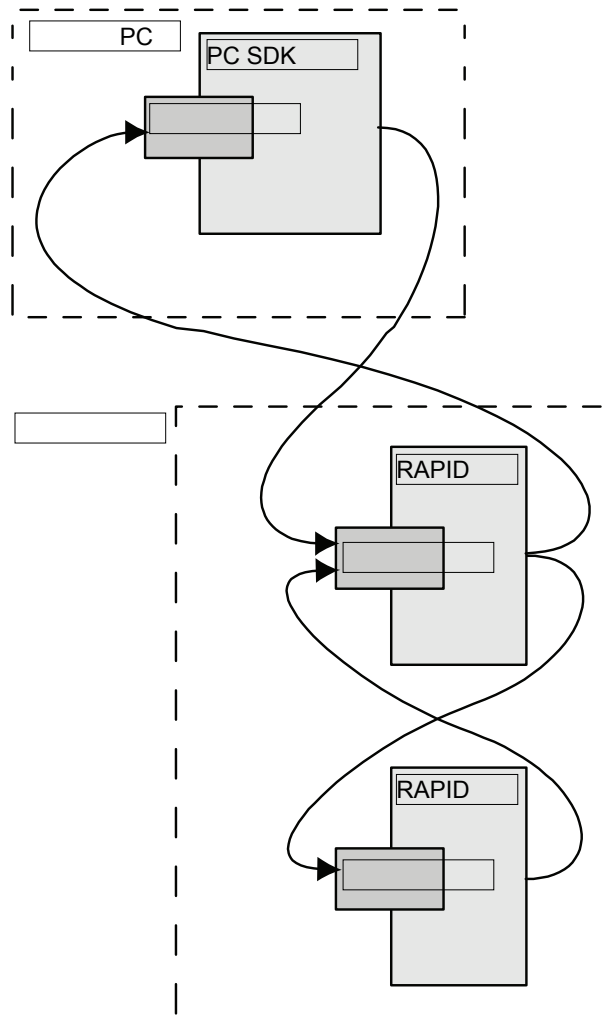
这是RAPID Message Queue的一般用法。[第276页的代码示例](#)用一个更详细的示例展示了其具体用法。

- 1 对于中断模式：接收方设置了一则软中断例程来读取一则消息和关联一次中断，因此系统会在出现一则新消息时调用该软中断例程。
对于同步模式：系统会通过一次等候或接下来执行的RMQReadWait指令来处理该消息。
- 2 发送方会在接收方任务中的队列查询相应的槽标识。
- 3 发送方会发送该消息。

8.5.2 RAPID消息队列行为

通信图

下图展示了可能出现在本系统中的各种发送方、接收方和队列。每个箭头都例举了一种向队列粘贴一则消息的途径。



en0700000430

创建一个PC SDK客户端

本手册只描述了如何用RAPID Message Queue来让一项RAPID task与其它RAPID tasks和PC SDK进行通信。至于如何在一个PC SDK客户端上设置相关通信，则请参见<http://developercenter.robotstudio.com>。

一则消息所能发送的内容

除以下内容外，一则消息中的数据可以属于RAPID中的任何数据类型：

- 非值
- 半值
- motsetdata

下一页继续

8 Communication

8.5.2 RAPID消息队列行为

续前页

一则消息中的数据可以是属于某种数据类型的一个数组。

允许使用用户定义的记录，但发送方和接收方都必须对该记录作出一模一样的声明。



提示

为了保持向后兼容性，一旦在发布的某款产品中使用了用户定义的记录，就请不要更改该记录。更好的做法是创建一份新的纪录，这样就既能接收旧应用的消息，也能接收新应用的消息。

队列名称

为RAPID task配置的队列名称就是带有前缀“RMQ_”的任务名称，比如RMQ_T_ROB1这样。指令RMQFindSlot会使用该名称。

队列处理

系统会按接收消息的顺序来处理队列中的消息，这种特性被称做FIFO，即“先进先出”。如果还在处理前一则消息时就接收到了新的消息，那么系统会把该新消息置于队列中。一旦第一则消息处理完毕，系统便会立即处理队列中的下一则消息。

队列模式

用系统参数*RMQ Mode*来定义相应的队列模式。默认行为为中断模式。

中断模式

中断模式下的消息处理情况取决于数据类型。系统只会处理已关联了其数据类型的消息。

必须为宜由接收方处理的每种数据类型设置一种循环中断。可在不同的中断中调用同一则软中断例程，即是说数据类型不止一种。

系统会抛弃数据类型未关联到中断上的消息，并仅在事件日志中留下一则警告消息。

接收对指令RMQSendWait的答复不会导致一次中断。无需设置中断便可接收这一答复。

同步模式

在同步模式下，该项任务会执行一条RMQReadWait指令来接收任何数据类型的一则消息。系统会将所有消息排队，然后按其抵达顺序处理。

若有一条等候RMQReadWait指令，则系统会立即处理该消息。

若没有等候RMQReadWait指令，则接下来执行的RMQReadWait指令将会处理该消息。

消息内容

RAPID Message Queue消息由包含接收方标识的一条标题和一则RAPID消息组成。该RAPID消息为一段优质打印的字符串，其数据类型名称（以及数组维度）后面是实际数据值。

RAPID消息示例：

```
"robtargt;[[930,0,1455],[1,0,0,0],[0,0,0,0],
[9E9,9E9,9E9,9E9,9E9,9E9]]"
"string;"A message string"
"msgrec;[100,200]"
"bool{2,2};[[TRUE,TRUE],[FALSE,FALSE]]"
```

下一页继续

未执行RAPID任务

即使当前并未在执行包含了某RAPID任务队列的RAPID任务，用户也或可向该队列粘贴消息。到再次执行相关RAPID任务为止都不会执行相应的中断。

消息大小限制

系统发送一则消息前会先计算其最大大小（其具体数据类型和维度的最大大小）。如果其大小超过了5000字节，那么系统将抛弃该消息，并产生一项错误；如果接收方是一个最大消息大小不到400字节的PC SDK客户端，那么发送方也会得到相同的错误。对拥有具体数据类型和维度的一则消息而言，系统要么始终都能发送该消息，要么始终无法发送该消息。

当收到一则消息（调用指令RMQGetMsgData）时，系统会计算其最大大小（其具体数据类型和维度的最大大小）。如果其大小超过了为该任务的队列配置的最大消息大小，那么系统将抛弃该消息，并记录一项错误。对拥有具体数据类型和维度的一则消息而言，系统要么始终都能接收该消息，要么始终无法接收该消息。

消息丢失

在中断模式下，系统会抛弃RAPID任务无法接收的任何消息。这些消息将会丢失，而事件日志中则会加入一条警告。

抛弃一则消息的部分原因：

- 接收任务不支持所发送的数据类型。
- 接收任务并未为所发送的数据类型设置中断，也没有任何RMQSendWait指令在等候该数据类型。
- 接收任务的中断队列已满

队列丢失

会在上电失败时清除该队列。

当丢失了RAPID task中的执行文本（比如将相应的程序指针移到主例程处）时，系统便会清空相应队列。

相关信息

队列与消息方面的更多信息请参见技术参考手册 - *RAPID*语言内核。

8.5.3 系统参数

关于系统参数

此处简述了RAPID Message Queue中的每个参数。更多信息请参见技术参考手册 - 系统参数中的各个参数。

类型Task

这些参数属于主题`Controller`下的类型`Task`。

参数	描述
RMQ Type	<p>可以拥有下列数值之一：</p> <ul style="list-style-type: none">• <i>None</i> – 就RAPID task来禁用与RAPID Message Queue之间的一切通信。• <i>Internal</i> – 启用从相关控制器上的其它任务处——而不是从外部客户端（FlexPendant示教器和各种PC应用）处——接收RAPID Message Queue消息。该任务仍能向外部客户端发送消息。• <i>Remote</i> – 就该项任务启用与RAPID Message Queue之间的通信，这既包括与相关控制器上的其它任务之间的通信，也包括与外部客户端（FlexPendant示教器和各种PC应用）之间的通信。 <p>默认值为<i>None</i>。</p>
RMQ Mode	<p>定义相关队列的模式。</p> <p>可以拥有下列数值之一：</p> <ul style="list-style-type: none">• <i>Interrupt</i> – 只有将一则软中断例程与一种指定的消息类型关联在一起后，系统才能接收该消息。• <i>Synchronous</i> – 只有在执行了一条RMQReadWait指令后，系统才能接收该消息。 <p>默认值为<i>Interrupt</i>。</p>
RMQ Max Message Size	<p>一则消息的最大数据大小（以字节计）。</p> <p>默认值为400。</p> <p>用户无法在RobotStudio中或FlexPendant上更改该数值，而只能通过编辑sys.cfg文件来更改该数值。其最大值为3000。</p>
RMQ Max No Of Messages	<p>队列中的最大消息数目。</p> <p>默认为5。</p> <p>用户无法在RobotStudio中或FlexPendant上更改该数值，而只能通过编辑sys.cfg文件来更改该数值。其最大值为10。</p>

8.5.4 RAPID组件

关于RAPID组件

此处简述了RAPID Message Queue中的每条指令、每则函数和每种数据类型。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各个参数。

指令：

指令	描述
RMQFindSlot	查找为某一RAPID task或Robot Application Builder客户端配置的队列的槽标识号。
RMQSendMessage	向为某一RAPID task或Robot Application Builder客户端配置的队列发送数据。
IRMQMessage	命令并启用某一具体数据类型的循环中断。
RMQGetMessage	从该项任务的队列中获取第一则消息。只有在 <i>RMQ Mode</i> 被定义为 <i>Interrupt</i> 的情况下才能使用该指令。
RMQGetMsgHeader	获取一则消息中的标题部分。
RMQGetMsgData	获取一则消息中的数据部分。
RMQSendWait	发送一则消息，然后等候答复。只有在 <i>RMQ Mode</i> 被定义为 <i>Interrupt</i> 的情况下才能使用该指令。
RMQReadWait	等候消息。只有在 <i>RMQ Mode</i> 被定义为 <i>Synchronous</i> 的情况下才能使用该指令。
RMQEmptyQueue	清空相应队列。

函数

功能	描述
RMQGetSlotName	在给定槽标识号（即给定rmqslot）情况下获取为某一RAPID task或Robot Application Builder客户端配置的队列的名称。

数据类型

数据类型	描述
rmqslot	某一RAPID task或Robot Application Builder客户端的槽标识。
rmqmessage	与RAPID Message Queue通信时用来保存数据的一则消息，其中包含了所发送的数据类型、发送方的槽标识和实际数据这些信息。 注意：rmqmessage是一种较大的数据类型。为该数据类型声明太多变量会导致内存问题。请尽量重复使用相同的rmqmessage变量。
rmqheader	rmqheader描述了相关消息，并可由RAPID程序加以读取。

8.5.5 代码示例

涉及RMQSendMessage和RMQGetMessage的示例

此例中的发送方创建了数据（x和y值），并将其发送给另一项任务。相关的接收任务获得了该信息，并将这些数据提取到名为data的变量中。

发送方

```
MODULE SenderMod
  RECORD msgrec
    num x;
    num y;
  ENDRECORD

  PROC main()
    VAR rmqslot destinationSlot;
    VAR msgrec data;
    VAR robtarget p_current;

    ! Connect to queue in other task
    RMQFindSlot destinationSlot "RMQ_OtherTask";

    ! Perform cycle
    WHILE TRUE DO
      ...
      p_current := CRobT(\Tool:=tool1 \Wobj:=wobj0);
      data.x := p_current.trans.x;
      data.y := p_current.trans.y;
      ! Send message
      RMQSendMessage destinationSlot, data;
      ...
    ENDWHILE
  ERROR
    IF ERRNO = ERR_RMQ_INVALID THEN
      WaitTime 1;
      ! Reconnect to queue in other task
      RMQFindSlot destinationSlot "RMQ_OtherTask";
      ! Avoid execution stop due to retry count exceed
      ResetRetryCount;
      RETRY;
    ELSIF ERRNO = ERR_RMQ_FULL THEN
      WaitTime 1;
      ! Avoid execution stop due to retry count exceed
      ResetRetryCount;
      RETRY;
    ENDIF
  ENDPROC
ENDMODULE
```

PC SDK客户端

```
public void RMQReceiveRecord()
```

下一页继续

```

{
    const string destination_slot = "RMQ_OtherTask";
    IpcQueue queue = Controller.Ipc.CreateQueue(destination_slot,
        16, Ipc.MaxMessageSize);

    // Till application is closed
    while (uiclose)
    {
        IpcMessage message = new IpcMessage();
        IpcReturntype retValue = IpcReturntype.Timeout;

        retValue = queue.Receive(1000, message);
        if (IpcReturntype.OK == retValue)
        {
            // PCSDK App will receive following record
            // RECORD msgrec
            // num x;
            // num y;
            // ENDRECORD

            // num data type in RAPID is 3 bytes long, hence will receive
            // 6 bytes for x and y
            // first byte do left shift by 16,
            // second byte do left shift by 8 and OR all three byte to
            // get x
            // do similar for y
            Int32 x = (message.Data[0] << 16) | (message.Data[1] << 8)
                | message.Data[2];
            Int32 y = (message.Data[3] << 16) | (message.Data[4] << 8)
                | message.Data[5];

            // Display x and y
        }
    }

    if (Controller.Ipc.Exists(destination_slot))
        Controller.Ipc.DeleteQueue(Controller.Ipc.GetQueueId(destination_slot));
}

```

涉及RMQSendWait的示例

此例中的RAPID程序发送了一则消息，然后等候答复，最后通过获取相应的答复消息来继续执行过程。

```

MODULE SendAndReceiveMod
    VAR rmqslot destinationSlot;
    VAR rmqmessage recmsg;
    VAR string send_data := "How many units should be produced?";
    VAR num receive_data;

    PROC main()
        ! Connect to queue in other task
        RMQFindSlot destinationSlot "RMQ_OtherTask";

```

下一页继续

```
! Send message and wait for the answer
RMQSendWait destinationSlot, send_data, recmsg, receive_data
    \Timeout:=30;

! Handle the received data
RMQGetMsgData recmsg, receive_data;
TPWrite "Units to produce: " \Num:=receive_data;

ERROR
IF ERRNO = ERR_RMQ_INVALID THEN
    WaitTime 1;
    ! Reconnect to queue in other task
    RMQFindSlot destinationSlot "RMQ_OtherTask";
    ! Avoid execution stop due to retry count exceed
    ResetRetryCount;
    RETRY;
ELSIF ERRNO = ERR_RMQ_FULL THEN
    WaitTime 1;
    ! Avoid execution stop due to retry count exceed
    ResetRetryCount;
    RETRY;
ELSEIF ERRNO = ERR_RMQ_TIMEOUT THEN
    ! Avoid execution stop due to retry count exceed
    ResetRetyCount;
    RETRY;
ENDIF
ENDPROC
ENDMODULE
```

涉及RMQReceiveSend的示例

```
public void RMQReceiveSend()
{
    const string destination_slot = "RMQ_OtherTask";
    IpcQueue queue = Controller.Ipc.CreateQueue(destination_slot,
        16, Ipc.MaxMessageSize);

    // Till application is closed
    while (uiclose)
    {
        IpcMessage message = new IpcMessage();
        IpcReturnType retValue = IpcReturnType.Timeout;

        retValue = queue.Receive(1000, message);
        if (IpcReturnType.OK == retValue)
        {
            // Received message "How many units should be produced?"
            if (message.ToString() == "How many units should be
                produced?")
            {
                Int32 UnitsToProduce = 100;
            }
        }
    }
}
```

```
// num data type in Rapid is 3 bytes long, hence will
// send 3 bytes to Rapid Module
byte[] @bytes = new byte[3];
bytes[0] = (byte)(UnitsToProduce >> 16);
bytes[1] = (byte)(UnitsToProduce >> 8);
bytes[2] = (byte)UnitsToProduce;

// Send UnitsToProduce to Rapid Module
message.SetData(@bytes);
queue.Send(message);
}
}
}

if (Controller.Ipc.Exists(destination_slot))
Controller.Ipc.DeleteQueue(Controller.Ipc.GetQueueId(destination_slot));
}
```

此页刻意留白

9 Engineering tools

9.1 Multitasking [623-1]

9.1.1 Multitasking介绍

目的

选项*Multitasking*的作用是能够同时执行多段程序。

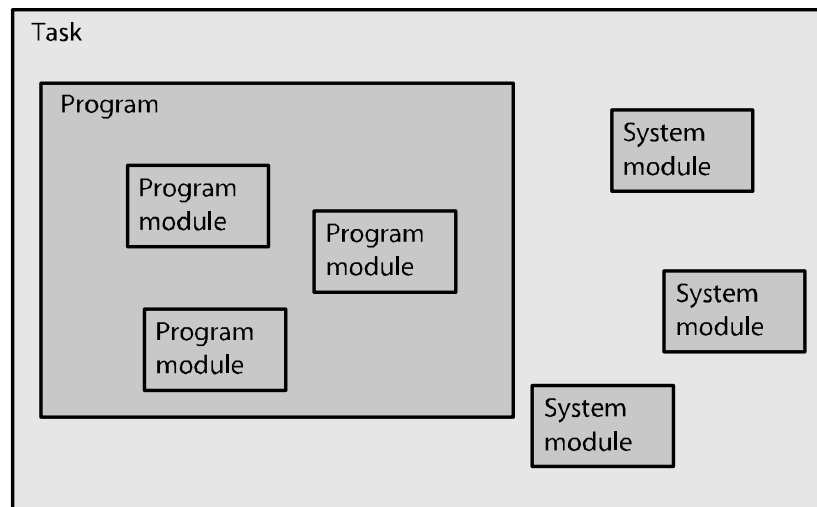
让各项应用与主程序并行运行的示例：

- 即使主程序已经停止，也仍继续进行信号监控。有时这会接手PLC的工作，不过其响应时间与PLC并不相符。
- 相关机器人正在工作时来自FlexPendant示教器的操作员输入项。
- 外部设备的控制和激活 / 停用。

基本说明

同一时间最多运行20项任务。

每项任务都由一段程序（含若干程序模块）和若干个系统模块组成。这些模块是各项任务中的本地模块。



en0300000517

变量和常量都是各项任务中的本地量，但持续变量并不是本地量。

每项任务都有自身的软中断处理事项，且只有其自身的任务系统状态才能触发事件例程。

其中包括

您可通过RobotWare选项Multitasking来访问：

- 最多能并行运行20段程序（每项任务运行一段）。
- 系统参数：类型*Task*及其所有参数。
- 数据类型：*taskid*、*syncident*和*tasks*。
- 指令：*WaitSyncTask*。

下一页继续

9 Engineering tools

9.1.1 Multitasking介绍

续前页

- 函数：TestAndSet、TaskRunMec和TaskRunRob。



注意

没有选项Multitasking也能使用TestAndSet、TaskRunMec和TaskRunRob，但搭配Multitasking时会大大提升它们的作用。

基本方法

这是设置Multitasking的基本方式。更多信息请参见[第285页的设置任务的调试策略](#)和[第284页的RAPID组件](#)。

- 1 定义您需要的任务。
- 2 写入每项任务的RAPID代码。
- 3 指定载入各项任务的模块。

9.1.2 系统参数

关于系统参数

此处简述了Multitasking中的每个参数。更多信息请参见技术参考手册 - 系统参数中的各个参数。

Task

这些参数属于主题`Controller`下的类型`Task`。

参数	描述
Task	<p>任务名称。</p> <p>注意该任务的名称必须是唯一的，这意味着不能是相关机械单元的名称，相应的RAPID程序中也不能有相同名称的变量。</p> <p>请注意，若在配置编辑器中编辑该任务条目并更改任务名称，则系统会移除旧任务并添加一项新任务。这意味着一旦做出此类改动，该任务中的所有程序或模块就会在重启后消失。</p>
Task in foreground	<p>用于设置各项任务的优先级。</p> <p><i>Task in foreground</i>包含了宜在该任务前台运行的任务名称。这意味着只有在前台任务程序空闲时，系统才会执行该参数设置的任务程序。</p> <p>如果<i>Task in foreground</i>被设置成某项任务的空字符串，那么它就会在最高等级上运行。</p>
Type	<p>控制启动 / 停止和系统重启行为：</p> <ul style="list-style-type: none"> NORMAL – 手动启动和停止该任务程序（比如通过FlexPendant示教器来启动和停止）。紧急停止时系统会停止该任务。 STATIC – 重启时该任务程序会从所处位置继续执行。不论是FlexPendant示教器还是紧急停止都通常不会停止该任务程序。 SEMISTATIC – 重启时该任务程序会从起点处重启。不论是FlexPendant示教器还是紧急停止都通常不会停止该任务程序。 <p>凡是控制着机械单元的任务，其类型都必须是NORMAL才行。</p>
Main entry	该任务程序的启动例程的名称。
Check unresolved references	如果系统在链接有一个模块的情况下接受了相关程序中的未决引用项，那么就宜把该参数设置成NO，反之则设置成YES。
TrustLevel	<p><i>TrustLevel</i>定义了一项STATIC或SEMISTATIC任务被停止（比如因错误而停止）时的系统行为：</p> <ul style="list-style-type: none"> SysFail – 若该项任务的程序停止，则系统会被设置成SYS_FAIL，从而导致所有“正常”任务的相应程序全部停止（不过系统会尽量继续执行STATIC和SEMISTATIC任务）。此时既不能进行任何点动，也不能启动任何程序。用户需要重启一次。 SysHalt – 若该项任务的程序停止，则系统会停止所有NORMAL任务的相应程序。如果设置了“电机开启”，那么就可以进行点动，但不能启动任何程序。用户需要重启一次。 SysStop – 若该项任务的程序停止，则系统会停止所有NORMAL任务的相应程序，但可以重启这些任务。用户可以点进行点动。 NoSafety – 系统仅会停止该项任务的程序。
MotionTask	<p>指明该任务程序能否用RAPID移动指令来控制机器人的移动行为。</p> <p>除非使用了选项MultiMove，否则就只能把一项任务的<i>MotionTask</i>设置成YES。</p>

9.1.3 RAPID组件

数据类型

此处简述了Multitasking中的每种数据类型。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各种数据类型。

数据类型	描述
taskid	taskid会识别本系统中的可用任务。 系统参数Task定义了这一标识，而RAPID程序则无法定义该标识。不过声明例程时可将数据类型taskid作为一个参数。 代码方面的示例请参见第300页的taskid。
syncident	syncident的作用是在使用指令WaitSyncTask时识别相关程序中的等候点。 所有任务程序中的syncident变量都必须具有同一个名称。 代码方面的示例请参见第294页的WaitSyncTask示例。
tasks	数据类型tasks的一个变量中包含了将通过指令WaitSyncTask实现同步的各项任务的名称。 代码方面的示例请参见第294页的WaitSyncTask示例。

指令

此处简述了Multitasking中的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
WaitSyncTask	WaitSyncTask的作用是在相关程序的某个特殊点处实现若干任务程序的同步。 一条WaitSyncTask指令会延缓程序的执行过程，并等候其它任务程序。 当所有任务程序都抵达指定点时，该程序才会继续执行。 代码方面的示例请参见第294页的WaitSyncTask示例。

函数

此处简述了Multitasking中的每则函数。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各则函数。

功能	描述
TestAndSet	TestAndSet的作用，是和一个布尔（boolean）旗标共同确保同一时间只有一段任务程序在使用某一具体的RAPID代码区域或系统资源。 代码方面的示例请参见第298页的涉及旗标的示例与TestAndSet。
TaskRunMec	检查该任务程序是否控制了任何机械单元（机器人或其它单元）。 代码方面的示例请参见第299页的测试任务是否控制着机械单元。
TaskRunRob	检查该任务程序是否控制了任何带有TCP的机器人。 代码方面的示例请参见第299页的测试任务是否控制着机械单元。

9.1.4 任务配置

9.1.4.1 设置任务的调试策略



提示

下列指令展示了如何安全地进行更新。将参数Type设置成NORMAL、将TrustLevel设置成NoSafety后，用户便既能更轻松地测试相关任务程序，又能更容易地校正任何可能发生的错误。

如果您确信您引入的代码是正确的，那么您可以跳过“更改Type和TrustLevel的数值”这一部分。如果您不更改任何系统参数，那么或许就不必使用任何重启模式。

设置任务

向您的系统添加一项新任务时请遵循这一指令。

	操作
1	为主题Controller下的系统参数类型Task添加一条实例，从而定义新的任务。
2	将参数Type设置成NORMAL。 这将使用户更容易创建和测试该任务中的各个模块。
3	创建宜用于该任务中的模块（不论是用FlexPendant示教器创建还是离线创建），然后保存这些模块。
4	在主题Controller和类型Automatic loading of Modules的系统参数中指定宜预载入相关新任务中的所有模块。 用户可以稍后载入NORMAL任务的此类模块，但必须预先载入STATIC或SEMISTATIC任务的此类模块。
5	停止相关控制器。
6	在“电机开启”状态下测试和调试这些模块，直至相关功能的效果令人满意为止。
7	将参数Type和TrustLevel改为所需值（比如SEMISTATIC和SysFail）。
8	重启系统。

更改任务程序

在编辑现有任务的一段程序时，如果将Type设置成STATIC或SEMISTATIC，那么就请遵循这一指令。

	操作
1	将系统参数TrustLevel设置成NoSafety。 这将使用户能够更改和测试该任务中的各个模块。
2	如果需要更改这一系统参数，则请重启相应的控制器。
3	在FlexPendant示教器的ABB菜单上启动控制面板，然后轻击FlexPendant示教器和任务面板设定。选择所有任务，然后轻击OK。
4	在“快速设置”菜单中选择要手动启停的任务。具体请参见第289页的选择用“启动”按钮启动哪项任务。
5	按下“停止”按钮来停止所选的“静态”和“半静态”任务。
6	启动程序编辑器。 此时也能编辑STATIC和SEMISTATIC任务了。

下一页继续

9 Engineering tools

9.1.4.1 设置任务的调试策略

续前页

	操作
7	更改、测试和保存这些模块。
8	再次启动控制面板，然后打开任务面板设定。选择仅正常任务，然后轻击OK。
9	将参数 <i>TrustLevel</i> 改回所需值（比如SysFail）。
10	重启系统。

9.1.4.2 优先级

如何排列工作优先级。

默认行为是所有任务程序都以相同的优先级按“循环赛”方式运行。

用户或可通过一项任务的后台设置来更改另一项任务的优先级，这样一来，只有当前台任务程序处于闲置状态（比如在等候某起事件）时，后台任务才会执行自己的程序。还有另一种情况，即前台任务程序已执行了一条移动指令，使得该前台任务必需等到相关机器人移动为止，于是系统便在此期间执行后台任务程序。

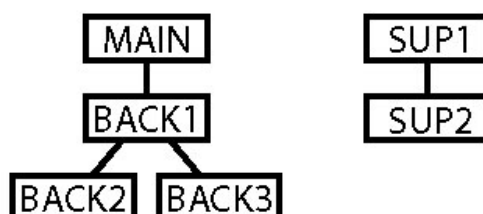
若要在其它任务的后台设置一项任务，则请使用参数*Task in foreground*。

优先级示例

使用了6项任务，它们的*Task in foreground*设置如下表所示。

任务名称	前台任务
MAIN	
BACK1	MAIN
BACK2	BACK1
BACK3	BACK1
SUP1	
SUP2	SUP1

此后的优先级结构将如下所示：

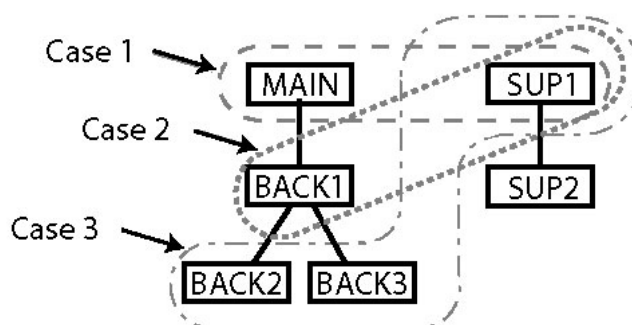


en0300000451

任务MAIN和SUP1的程序将轮流逐一执行每条指令（下图中的案例1）。

如果MAIN任务程序处于闲置状态，那么任务BACK1和SUP1的程序将轮流逐一执行每条指令（下图中的案例2）。

如果MAIN和BACK1任务程序都处于闲置状态，那么任务BACK2、BACK3和SUP1的程序将轮流逐一执行每条指令（下图中的案例3）。



en0300000479

9.1.4.3 任务面板设定

任务面板设定的作用

默认行为是“启动”和“停止”按钮仅会启动和停止NORMAL任务”。您可在“任务选择面板”中选择要启动和停止哪项NORMAL任务。具体请参见[第289页的选择用“启动”按钮启动哪项任务](#)。

用户可在“任务面板设定”中更改默认行为，从而得以用“启动”和“停止”按钮来步进、启动和停止STATIC和SEMISTATIC任务。不过只有当这些任务的*TrustLevel*被设置成NoSafety时，才能且仅能在手动模式下启动和停止这些任务。

允许在任务面板上选择STATIC和SEMISTATIC任务

以下无返回值程序详述了如何才能在任务面板上选择STATIC和SEMISTATIC。

	操作
1	在ABB菜单上轻击控制面板，再轻击FlexPendant示教器，然后轻击任务面板设定。
2	选择信任等级为不安全的所有任务（正常 / 静态 / 准静态），然后轻击OK。

9.1.4.4 选择用“启动”按钮启动哪项任务

后台

默认行为是“当按下‘启动’按钮时，所有NORMAL任务的程序都会同时启动”。不过并非所有的NORMAL任务程序都需要同时运行。用户可以选择按下“启动”按钮时将会启动的NORMAL任务程序。

如果在任务面板设定中选择了所有任务，那么只要是*TrustLevel*被设置成NoSafety的STATIC和SEMISTATIC任务，用户就能用“启动”按钮启动其程序、用“FWD”按钮向前步进其程序、用“BWD”按钮向后步进其程序以及用“停止”按钮停止其程序。

如果任务面板设定被设置成仅正常任务，那么所有的STATIC和SEMISTATIC任务会呈灰色，且无法在“快速设置”菜单的任务面板上选中（请参见操作员手册 - 带*FlexPendant*的IRC5的节“快速设置”菜单）。如果按下启动按钮，那么所有的STATIC和SEMISTATIC任务都会启动。

如果任务面板设定被设置成所有任务，那么用户可以在任务面板上选择*TrustLevel/NoSafety*的STATIC和SEMISTATIC任务。用户可停止、步进和启动所有选中的STATIC和SEMISTATIC任务。

即使未在任务面板上选择某项STATIC或SEMISTATIC任务，用户也仍可执行这项任务。不过用户无法执行未选中的NORMAL任务。

“静态”和“准静态”任务的“执行模式”始终都是连续的。“快速设置”菜单中的“执行模式”仅适用于“正常”任务（请参见操作员手册 - 带*FlexPendant*的IRC5的节“快速设置”菜单）。

这只会手动模式下发挥作用；用户无法在自动模式下启动、步进或停止任何“静态”或“准静态”任务。

任务面板设定

若要启动任务面板设定，则轻击ABB菜单，然后轻击控制面板、*FlexPendant*示教器和任务面板设定。

选择任务

使用该程序来选择将用“启动”按钮启动的任务。

	操作
1	将相关控制器设置成手动模式。
2	在 <i>FlexPendant</i> 示教器上轻击“快速设置”菜单，然后轻击任务面板按钮来显示所有任务。 如果任务面板设定被设置成仅正常任务，那么所有的STATIC和SEMISTATIC任务都会呈现出灰色的无法选中状态。 如果任务面板设定被设置成所有任务，那么用户可以选择 <i>TrustLevel/NoSafety</i> 的STATIC和SEMISTATIC任务，而“信任等级”被设置成其它数值的STATIC和SEMISTATIC任务则会呈现出灰色的无法选中状态。
3	若是宜用“启动”按钮来启动程序的任务，则选中其复选框。

在手动模式下重置调试设定

用该无返回值程序来恢复手动模式下的正常执行过程。

	操作
1	在任务面板设定中选择仅正常任务。

下一页继续

	操作
2	按下“启动”按钮。 所有STATIC和SEMISTATIC都将继续执行，“停止”按钮或紧急停止都无法停住它们。

切换为自动模式

当切换为自动模式时，系统会取消在任务面板上选中的所有STATIC和SEMISTATIC任务。已停止的“静态”和“半静态”任务将在下一次按下“启动”、“FWD”或“BWD”按钮之一时启动，然后继续向前执行。“停止”按钮或紧急停止均无法停住这些任务。

至于在任务面板上取消选中的NORMAL任务会怎么样，则取决于主题*Controller*下类型为*Auto Condition Reset*的系统参数*Reset*。如果*Reset*被设置成Yes，那么按下“启动”按钮就会选中并启动任务面板上的所有NORMAL任务；如果*Reset*被设置成No，那么“启动”按钮就只能启动在任务面板上选中的NORMAL任务。



注意

请注意，更改系统参数*Reset*的值会影响到所有调试重设定（比如速度超驰和仿真I/O）。更多信息请参见技术参考手册 - 系统参数的节*Auto Condition Reset*。

重启控制器

如果重启相关控制器，那么系统将保留所有“正常”任务的状态，并取消相关任务面板上所有选中的“静态”和“半静态”任务。当控制器启动时，系统会启动所有“静态”和“半静态”任务，然后连续执行这些任务。

在同步模式下取消选中任务

如果某项任务处在同步模式下（即是说程序指针位于SyncMoveOn与SyncMoveOff之间），那么用户可取消该任务的选中状态，但不能重新选择该任务。在同步终止前都不能选择该任务。如果继续执行下去，那么同步过程将最终按其它任务——而不是取消选中的任务——的需要而终止。即使将相应的程序指针移到主例程或某一例程处，也无法按取消选中任务的需要来终止同步。

在系统参数*Reset*被设置成Yes的情况下，如果被取消选中的任务正处在同步模式下，那么任何改为自动模式的尝试都将失败。改为自动模式理应会使所有NORMAL任务都被选中，所以若无法实现这一点，就无法改为自动模式。

9.1.5 各项任务间的通信

9.1.5.1 永久变量

关于永久变量

若要在各项任务之间共享数据，则请使用永久变量。

永久变量是声明了该变量的所有任务的一个全局变量。必须在所有任务中为相关永久变量声明相同的类型和大小（数组维度），否则将会发生执行时间错误。

其足以为一项任务中的永久变量指定一个初始值。如果在若干项任务中指定了初始值，那么系统仅会使用最先载入的模块的初始值。



提示

在保存一段程序时，系统会把某个永久变量的当前值作为日后的初始值。如果不需要这种设定，那么就在通信后直接重置相应的永久变量。

涉及永久变量的示例

此例中的两项任务都访问了永久变量startsync和stringtosend，所以可将这些变量用于相关任务程序之间的通信。

主任务程序：

```
MODULE module1
  PERS bool startsync:=FALSE;
  PERS string stringtosend:=" ";
  PROC main()
    stringtosend:="this is a test";
    startsync:= TRUE
  ENDPROC
ENDMODULE
```

后台任务程序：

```
MODULE module2
  PERS bool startsync;
  PERS string stringtosend;
  PROC main()
    WaitUntil startsync;
    IF stringtosend = "this is a test" THEN
      ...
    ENDIF
    !reset persistent variables
    startsync:=FALSE;
    stringtosend:=" ";
  ENDPROC
ENDMODULE
```

公用数据模块

当在若干项任务中使用永久变量时，最好在所有这些任务中都加以声明。而为了避免类型错误或忘了在某处声明，最好的做法就是在一个系统模块中声明所有公用变量，然后将该系统模块载入所有需要这些变量的任务中。

9.1.5.2 等候其它任务

两种技巧

某些应用会在独立于其它任务情况下执行任务程序，但各任务程序通常还是需要了解其它任务正处在何种状态的。

可让某一任务程序等候其它任务程序，具体来说，要么设置一个其它任务程序可以调用的永久变量，要么设置一个可让其它任务程序关联到中断上的信号。

轮询

这是让任务程序等候其它任务程序的最简单方式，但却是执行最慢的方式。搭配指令WaitUntil或WHILE来使用永久变量。

如果使用了指令WaitUntil，那么系统将每100毫秒实施一次内部轮询。



小心

轮询频率不要超过100毫秒一次。如果某环路的轮询没有等候指令，那么就会导致过载，进而丧失与FlexPendant示教器之间的联系。

轮询示例

主任务程序：

```
MODULE module1
  PERS bool startsync:=FALSE;
  PROC main()
    startsync:= TRUE;
    ...
  ENDPROC
ENDMODULE
```

后台任务程序：

```
MODULE module2
  PERS bool startsync:=FALSE;
  PROC main()

    WaitUntil startsync;
    ! This is the point where the execution
    ! continues after startsync is set to TRUE
    ...
  ENDPROC
ENDMODULE
```

中断

如果在某一任务程序中设置一个信号，并使用另一段任务程序中的一次中断，那么就既能快速响应，又没有轮询带来的工作负载。

其缺点是“必须把在此次中断后执行的代码放置在一则软中断例程中”。

中断示例**主任务程序：**

```
MODULE module1
  PROC main()
    SetDO do1,1;
    ...
  ENDPROC
ENDMODULE
```

后台任务程序：

```
MODULE module2
  VAR intnum intnol;

  PROC main()
    CONNECT intnol WITH wait_trap;
    ISignalDO do1, 1, intnol;
    WHILE TRUE DO
      WaitTime 10;
    ENDWHILE
  ENDPROC

  TRAP wait_trap
    ! This is the point where the execution
    ! continues after do1 is set in main task
    ...
    IDelete intnol;
  ENDTRAP
ENDMODULE
```

9.1.5.3 多项任务之间的同步

用WaitSyncTask实现同步

当各任务程序取决于彼此时，同步就会发挥作用。除非所有任务程序都抵达了各自程序代码中的同步点，否则这些任务程序都不会继续执行到其程序代码的同步点以外。

指令WaitSyncTask的作用是同步各任务程序。除非所有任务程序都抵达了同一WaitSyncTask指令处，否则这些任务程序都不会继续执行。

WaitSyncTask示例

在此例中，当相关的后台任务程序在计算下一个对象的位置时，主任务程序正在处理涉及当前对象的机器人工作。

该后台任务程序可能不得不等候操作员输入项或I/O信号，但在计算出相应的新位置前，主任务程序不会用下一个对象继续执行。与此类似，除非已用一个对象执行了主任务程序，且该主任务程序已准备好接收新值，否则相应的后台任务程序不得开始下一次计算。

主任务程序：

```
MODULE module1
  PERS pos object_position:= [0,0,0];
  PERS tasks task_list{2} := [{"MAIN"}, {"BACK1"}];
  VAR syncident sync1;

  PROC main()
    VAR pos position;
    WHILE TRUE DO
      !Wait for calculation of next object_position
      WaitSyncTask sync1, task_list;
      position:=object_position;
      !Call routine to handle object
      handle_object(position);
    ENDWHILE
  ENDPROC

  PROC handle_object(pos position)
    ...
  ENDPROC
ENDMODULE
```

后台任务程序：

```
MODULE module2
  PERS pos object_position:= [0,0,0];
  PERS tasks task_list{2} := [{"MAIN"}, {"BACK1"}];
  VAR syncident sync1;

  PROC main()
    WHILE TRUE DO
      !Call routine to calculate object_position
      calculate_position;
    ENDWHILE
  ENDPROC
ENDMODULE
```

```
        !Wait for handling of current object
        WaitSyncTask sync1, task_list;
    ENDWHILE
ENDPROC

PROC calculate_position()
    ...
    object_position:= ...
ENDPROC
ENDMODULE
```

9.1.5.4 使用调度程序

什么是调度程序？

可用一个数字信号来指明宜于何时让另一项任务来开展工作。不过该信号无法包含工作内容方面的信息。

用户可用一段调度程序——而不是使用一个信号——来决定调用哪一则例程。调度程序可以是一个永久字符串变量，其中包含了将在另一项任务中执行的例程之名称。

调度程序示例

在此例中，我们将`routine_string`设置成相应的例程名称，然后设置成从5进行到1，从而让主任务程序调用了相关后台任务中的多则例程。通过这种方式，主任务程序可初始化成以下状态：相关后台任务程序宜首先执行例程`clean_gun`，然后执行`routine1`。

主任务程序：

```
MODULE module1
  PERS string routine_string:="";

  PROC main()
    !Call clean_gun in background task
    routine_string:="clean_gun";
    SetDO do5,1;
    WaitDO do5,0;

    !Call routine1 in background task
    routine_string:="routine1";
    SetDO do5,1;
    WaitDO do5,0;
    ...
  ENDPROC
ENDMODULE
```

后台任务程序：

```
MODULE module2
  PERS string routine_string:="";

  PROC main()
    WaitDO do5,1;
    %routine_string%;
    SetDO do5,0;
  ENDPROC

  PROC clean_gun()
    ...
  ENDPROC

  PROC routine1()
    ...
  ENDPROC
```


ENDMODULE

9.1.6 其它编程问题

9.1.6.1 在各项任务之间共享资源

指明资源已被占用的旗标

所有任务都能使用FlexPendant示教器、文件系统和I / O信号等系统资源，不过若有多段任务程序在使用同一资源，那么请确保它们是在轮流使用该资源，而不是在同时使用该资源。

若要避免两段任务程序同时使用同一资源，就用一个旗标来指明该资源已被使用。当任务程序正在使用该资源时，用户可将一个布尔变量设置成“真（true）”。

若要加快这一处理过程，则请使用指令TestAndSet。该指令将首先测试相关旗标，如果此旗标为“假（false）”，那么该指令就会将此旗标设置成“真”，并返回“真”；否则就返回“假”。

涉及旗标的示例与TestAndSet

在此例中，有两段任务程序试图在FlexPendant示教器上各写入三行。如果未使用旗标，那么这些行就存在彼此混合的风险；而在使用了旗标后，最先执行TestAndSet指令的任务程序就会最先写入所有三行，而另一段任务程序则会等到相关旗标被设置成“假”为止，然后写入自己的所有三行。

主任务程序：

```
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;
```

后台任务程序：

```
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;
```

9.1.6.2 测试任务是否控制着机械单元

用于询问的两则函数

某些函数会检查相关任务程序是否已控制了任何机械单元 (TaskRunMec) 或控制了一台机器人 (TaskRunRob)。

如果相关任务程序控制了一台机器人或其它机械单元, 那么TaskRunMec将返回“真”; 如果相关任务程序控制了一台带TCP的机器人, 那么TaskRunRob仅会返回“真”。

使用MultiMove时TaskRunMec和TaskRunRob 会有所帮助。若拥有MultiMove, 您就能让若干项任务控制多个机械单元。具体请参见应用手册 - *MultiMove*。



注意

就控制了一台机器人的某项任务而言, 参数Type必须被设置成NORMAL, MotionTask必须被设置成YES。具体请参见第283页的系统参数。

涉及TaskRunMec和TaskRunRob的示例

此例中设置了外部设备的最大速度。如果相关任务程序控制了一台机器人, 那么外部设备的最大速度就会被设置成该机器人的最大速度。如果相关任务程序控制了一部非机器人的外部设备, 那么其最大速度就会被设置成5000毫米 / 秒。

```
IF TaskRunMec() THEN
  IF TaskRunRob() THEN
    !If task controls a robot
    MaxExtSpeed := MaxRobSpeed();
  ELSE
    !If task controls other mech unit than robot
    MaxExtSpeed := 5000;
  ENDIF
ENDIF
```

9.1.6.3 taskid

taskid的语法

一项任务始终有一个类型为taskid的预定义变量（由该任务的名称和后缀“Id”组成），比如MAIN任务的这一变量名称就是MAINId。

代码示例

在此例中，尽管有另一项任务执行了Save指令，但还是把模块PART_A保存在了任务BACK1中。

BACK1Id是一个类型为taskid的变量。系统会自动声明该变量。

```
Save \TaskRef:=BACK1Id, "PART_A"  
  \FilePath:="HOME:/DOORDIR/PART_A.MOD";
```

9.1.6.4 避免冗长环路

后台任务继续形成环路

继续正常执行一段任务程序。这意味着某段后台任务程序不但是有效的，还形成了一段永恒环路。如果该程序没有任何等候指令，那么所述后台任务就可能占用过多的计算机能力，并导致相关控制器无法处理其它任务。

示例

```
MODULE background_module
  PROC main()
    WaitTime 1;
    IF di1=1 THEN
      ...
    ENDIF
  ENDPROC
ENDMODULE
```

如果此例中没有等候指令且di1为0，那么该后台任务的环路就会在一无所获的情况下完全占用相关计算机的能力。

9.2 Sensor Interface [628-1]

9.2.1 Sensor Interface介绍

目的

选项Sensor Interface的作用是通过一条串行通道与外部传感器进行通信。

用户可用一揽子RAPID指令（这些指令能读取和写入传感器数据）来访问该串行通道。

有种中断特性可订阅传感器数据的变动。



提示

Sensor Interface提供的通讯被集成在焊缝跟踪和工艺参数适应性控制所用的电弧焊指令中，这些指令会处理通讯，并根据您的需要加以校正；反之，若采用Sensor Interface，则是由您自己进行处理。更多信息请参见应用手册 - *Arc and Arc Sensor* 和应用手册 - *Continuous Application Platform*。

其中包括

您可通过RobotWare选项Sensor Interface来访问：

- 用于关联一件传感器装置的指令：SenDevice。
- 根据相关串行传感器接口的输入来设置中断的指令：IVarValue。
- 当某件装置关联到相关的串行传感器接口上时，用于对该装置进行读取和写入的指令：ReadBlock、WriteBlock和WriteVar。
- 当某件装置关联到相关的串行传感器接口上时，用于读取该装置的函数：ReadVar。

基本方法

这是Sensor Interface的基本用法。

- 1 配置相关传感器。具体请参见[第304页的通过串行通道来配置传感器](#)。
- 2 根据相关传感器的输入，运用RAPID代码中的中断来进行调节。具体示例请参见[第308页的中断焊接以调节设定](#)。

限制

只能用指令ArcL、ArcC、CapL和CapC来使用带IVarValue的中断。必须使用开关Track，即是说相关控制器必须配备RobotWare Arc或，Continuous Application Platform，并搭配上Optical Tracking或选项Weldguide。

9.2.2 配置传感器

9.2.2.1 关于这些传感器

受支持的传感器

传感器接口支持以下对象：

- 按照RTP1协议，通过串行通道来连接各个传感器。配置方面的信息请参见[第304页的通过串行通道来配置传感器](#)。
- 按照Servo-Robot Inc公司的RoboCom Light协议或ABB公司的LTAPP协议，把各个传感器连接到以太网上。配置方面的信息请参见[第305页的通过以太网通道来配置传感器](#)。

9.2.2.2 通过串行通道来配置传感器

概述

按照RTP1协议，Sensor Interface通过串行通道最多能与两个传感器进行通信。

系统参数

此处简述了配置一个传感器时所用的相关参数。关于这些参数的更多信息请参见技术参考手册 - 系统参数。

这些参数属于主题*Communication*下的类型*Transmission Protocol*。

参数	描述
Name	相关传输协议的名称。 就传感器而言，该名称必须以“:”结尾，比如“laser1:”或“swg:”这样。
Type	传输协议的类型。 对使用串行通道的传感器来说，它必须是“RTP1”。
Serial Port	将用于相关传感器的串行端口的名称。此处会引用类型为 <i>Serial Port</i> 的参数 <i>Name</i> 。 至于如何配置一个串行端口，则请参见技术参考手册 - 系统参数。

配置示例

此例展示了如何为一个传感器配置一项传输协议。

我们假设已经用名称“COM1”配置了一个串行端口。

Name	Type	Serial Port
laser1:	RTP1	COM1

9.2.2.3 通过以太网通道来配置传感器

概述

按照RoboCom Light协议版本E04（出自Servo-Robot Inc公司）或LTAPP协议（出自ABB公司），Sensor Interface通过以太网通道最多能与六个传感器进行通信。RoboCom Light是一种基于XML并使用了TCP/IP的协议。

该传感器会起到一台服务器的作用，而相关的机器人控制器则会起到一个客户端的作用，即是说该机器人控制器会初始化该传感器的连接状况。

RoboCom Light期望外部传感器一侧有TCP端口6344，而LTAPPTCP则期望有TCP端口5020。

系统参数

此处简述了配置一个传感器时所用的相关参数。关于这些参数的更多信息请参见技术参考手册 - 系统参数。

这些参数属于主题*Communication*下的类型*Transmission Protocol*。

参数	描述
Name	相关传输协议的名称。 就传感器而言，该名称必须以“:”结尾，比如“laser1:”或“swg:”这样。
Type	传输协议的类型。 用户必须为RoboCom Light配置相关的协议类型SOCKDEV，而LTAPPTCP的协议类型则是LTAPPTCP。
Serial Port	将用于相关传感器的串行端口的名称。此处会引用类型为 <i>Serial Port</i> 的参数 <i>Name</i> 。 至于如何配置一个串行端口，则请参见技术参考手册 - 系统参数。 基于IP的传输协议（即 <i>Type</i> 具有数值TCP/IP、SOCKDEV、LTAPPTCP或UDPUC）不使用 <i>Serial Port</i> ，且具有数值N/A。
Remote Address	该传感器的IP地址。此处会引用类型 <i>Remote Address</i> 。 至于如何配置Remote Address，则请参见技术参考手册 - 系统参数。

配置示例

这些例子展示了如何为一个传感器配置一项传输协议。

Name	Type	Serial Port	远程地址
laser2:	SOCKDEV	N/A	192.168.125.101
laser3:	LTAPPTCP	N/A	192.168.125.102

9 Engineering tools

9.2.3.1 RAPID组件

9.2.3 RAPID

9.2.3.1 RAPID组件

数据类型

没有针对*Sensor Interface*的数据类型。

指令

此处简述了*Sensor Interface*中的每条指令。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各条指令。

指令	描述
SenDevice	使用SenDevice来关联一件物理传感器装置。
IVarValue	IVarVal（中断变量值）的作用如下：通过相关串行传感器接口访问的某个变量若出现数值变化，则下令并启用一次中断。
ReadBlock	ReadBlock的作用是从关联到相关串行传感器接口的一件装置上读取一个数据块。系统会把该数据保存在一份文件中。 只能通过经串行通道连接的传感器（而不是经以太网连接的传感器）来使用ReadBlock。
WriteBlock	WriteBlock的作用是在关联到相关串行传感器接口的一件装置中写入一个数据块。系统从一份文件中抽取该数据。 只能通过经串行通道连接的传感器（而不是经以太网连接的传感器）来使用WriteBlock。
WriteVar	WriteVar的作用是在关联到相关串行传感器接口的一件装置中写入一个变量。

函数

此处简述了*Sensor Interface*中的每则函数。更多信息请参见技术参考手册 - *RAPID*指令、函数和数据类型中的各则函数。

功能	描述
ReadVar	ReadVar的作用是从关联到相关串行传感器接口的一件装置上写入一个变量。

模块

选项*Sensor Interface*包括了一个系统模块LTAPP__Variables。该模块包含了协议LTAPP中定义的相关变量号。系统会将其作为SHARED而自动载入，并让所有RAPID任务都能使用其所含变量（CONST数）。

注意！本机器人系统的目录HOME/LTC中有该模块的一个副本，但该副本不是载入的那个模块。

常数

名称	编号	读取 / 写入	描述
LTAPP__VERSION	1	R	识别相关传感器的软件版本的一个数值。
LTAPP__RESET	3	W	不论相关传感器当前处于何种状态，都一律将该传感器重置为初始状态。
LTAPP__PING	4	W	传感器返回了一项指明其状态的响应。

下一页继续

名称	编号	读取 / 写入	描述
LTAPP__CAMCHECK	5	W	启动相关传感器的摄像机检查。如果不能在相关链路协议指定的时间限制内完成检查，那么系统就将返回尚未就绪状态。
LTAPP__POWER_UP	6	RW	对相关传感器通电（1）或断电（0），并初始化相应的滤波器（通电时需要等候几秒钟！）
LTAPP__LASER_OFF	7	RW	打开（1）或关闭（0）相关激光束，并进行测量。
LTAPP__X	8	R	测得的X值，无符号字。变量单元决定了这些单元。
LTAPP__Y	9	R	测得的Y值，无符号字。变量单元决定了这些单元。
LTAPP__Z	10	R	测得的Z值，无符号字。变量单元决定了这些单元。
LTAPP__GAP	11	R	两张金属板之间的间隙。变量单元决定了相关单元。若无效，则为-32768。
LTAPP__MISMATCH	12	R	错配，无符号字。变量单元决定了相关单元。若无效，则为-32768。
LTAPP__AREA	13	R	焊缝区域（以毫米2为单位）。若无效，则为-32768。
LTAPP__THICKNESS	14	RW	相关传感器宜寻求的板厚度，LSB = 0.1毫米。
LTAPP__STEPPDIR	15	RW	相关关节的步进方向：向路径方向的左侧（1）或右侧（0）步进。
LTAPP__JOINT_NO	16	RW	设置或获取已激活关节的编号。
LTAPP__AGE	17	R	从获取轮廓起的时间（毫秒），无符号字。
LTAPP__ANGLE	18	R	相对于传感器坐标系Z方向的相关关节与法线之夹角 - 0.1度。
LTAPP__UNIT	19	RW	X、Y和Z间隙与错配的单位。0 = 0.1毫米，1 = 0.01毫米。
-	20	-	留作内部使用。
LTAPP__APM_P1	31	R	仅伺服机器人！适应性参数1
LTAPP__APM_P2	32	R	仅伺服机器人！适应性参数2
LTAPP__APM_P3	33	R	仅伺服机器人！适应性参数3
LTAPP__APM_P4	34	R	仅伺服机器人！适应性参数4
LTAPP__APM_P5	35	R	仅伺服机器人！适应性参数5
LTAPP__APM_P6	36	R	仅伺服机器人！适应性参数6
LTAPP__ROT_Y	51	R	围绕传感器Y轴测得的角度。
LTAPP__ROT_Z	52	R	围绕传感器Z轴A测得的角度。

9.2.4 示例

9.2.4.1 代码示例

中断焊接以调节设定

该段焊接程序示例使用了一个传感器。该传感器读取了相关间隙（以毫米为单位），而每当来自该传感器的这一数值发生变化时，就会发生一次中断，然后系统使用来自该传感器的新值来决定电压、馈线和速度方面的正确设定。

```
LOCAL PERS num adptVlt{8}:=
    [1,1.2,1.4,1.6,1.8,2,2.2,2.5];
LOCAL PERS num adptWfd{8}:=
    [2,2.2,2.4,2.6,2.8,3,3.2,3.5];
LOCAL PERS num adptSpd{8}:=
    [10,12,14,16,18,20,22,25];
LOCAL CONST num GAP_VARIABLE_NO:=11;
PERS num gap_value:=0;
PERS trackdata track:=[0,FALSE,150,[0,0,0,0,0,0,0,0],
    [3,1,5,200,0,0,0]];
VAR intnum IntAdap;

PROC main()
    ! Setup the interrupt. The trap routine AdapTrap 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 gap_value variable.
    CONNECT IntAdap WITH AdapTrap;
    IVarValue "laser1:", GAP_VARIABLE_NO, gap_value, IntAdap;

    ! Start welding
    ArcLStart p1,v100,adaptSm,adaptWd,fine, tool\j\Track:=track;
    ArcLEnd p2,v100,adaptSm,adaptWd,fine, tool\j\Track:=track;
ENDPROC

TRAP AdapTrap
    VAR num ArrInd;
    ! Scale the raw gap value received
    ArrInd:=ArrIndx(gap_value);

    ! Update active weld data variable adaptWd with new data from
    the predefined parameter arrays.
    ! The scaled gap value is used as index in the voltage, wirefeed
    and speed arrays.
    adaptWd.weld_voltage:=adptVlt{ArrInd};
    adaptWd.weld_wirefeed:=adptWfd{ArrInd};
    adaptWd.weld_speed:=adptSpd{ArrInd};

    ! Request a refresh of welding parameters using the new data in
    adaptWd
    ArcRefresh;
ENDTRAP
```

下一页继续

```
FUNC ArrIndx(num value)
  IF value < 0.5 THEN RETURN 1;
  ELSEIF value < 1.0 THEN RETURN 2;
  ELSEIF value < 1.5 THEN RETURN 3;
  ELSEIF value < 2.0 THEN RETURN 4;
  ELSEIF value < 2.5 THEN RETURN 5;
  ELSEIF value < 3.0 THEN RETURN 6;
  ELSEIF value < 3.5 THEN RETURN 7;
  ELSE RETURN 8;
ENDIF
ENDFUNC
```

读取来自传感器的位置

此例中不但打开了相关的传感器，还读取了该传感器中的数据。

```
! Define variable numbers
CONST num SensorOn := 6;
CONST num YCoord := 9;
CONST num ZCoord := 10;

! Define the transformation matrix
CONST pose SensorMatrix := [[100,0,0],[1,0,0,0]];

VAR pos SensorPos;
VAR pos RobotPos;

! Request start of sensor measurements
WriteVar SensorOn, 1;

! Read a Cartesian position from the sensor
SensorPos.x := 0;
SensorPos.y := ReadVar (YCoord);
SensorPos.z := ReadVar (ZCoord);

! Stop sensor
WriteVar SensorOn, 0;

! Convert to robot coordinates
RobotPos := PoseVect(SensorMatrix, SensorPos);
```

9.3 Externally Guided Motion [689-1]

9.3.1 EGM介绍

9.3.1.1 概述

目的

*Externally Guided Motion (EGM)*提供了两种不同特性：

- *EGM Position Guidance*:
相关机器人不会沿RAPID中的编程路径移动，而是沿某件外部装置所生成的路径移动。
- *EGM Path Correction*:
用某件外部装置提供的测量值来修改 / 校正所编写的机器人路径。

EGM Position Guidance

*EGM Position Guidance*的作用是用一件外部装置来生成一台或多台机器人所用的位置数据。这些机器人将会移到给定位置处。

一些应用示例：

- 将一个对象（比如一扇车门或车窗）放在某个外部传感器给出的位置（比如车体）处。
- 仓中取物。从一个料仓中捡拾对象（该料仓使用了一个外部传感器来识别这一对象及其位置）。

EGM Path Correction

*EGM Path Correction*的作用是用安装在装置上的外部机器人来生成一件外部装置来生成一台或多台机器人所用的路径校正数据。后一类机器人将会沿校正后的路径（即添加了所测校正值的相应编程路径）。

一些应用示例：

- 焊缝跟踪
- 跟踪在已知路径附近移动的对象。

其中包括

您可通过RobotWare选项*Externally Guided Motion*来访问：

- 用于设置、激活和重置EGM Position Guidance的指令。
- 用于设置、激活和重置EGM Path Correction的指令。
- 用于初始化EGM Position Guidance移动和停止此类移动的指令。
- 用于实现EGM Path Correction移动的指令。
- 用于检索当前EGM状态的一则函数。
- 用于配置EGM和设置默认值的系统参数。

限制

EGM Position Guidance的限制

- 由于EGM Position Guidance不包含插补器功能，因此用户无法用EGM Position Guidance来实现直线移动。相关机器人的实际路径将取决于该机器人的配置、启动位置和所生成的位置数据。
- EGM Position Guidance不支持MultiMove。
- 如果某个工件正在移动，那么用户就无法用EGM Position Guidance来指引该工件中的某个机械单元。
- 如果相关机器人陷入了某个奇异点附近（即说两根机器人轴近乎平行），那么系统会停止该机器人的移动并产生一则错误消息。此时唯一的办法就是以点动方式让该机器人脱离奇异点。
-

EGM Path Correction的限制

- 该外部装置必须安装有机器人。
- 只能在相关的路径坐标系上实施校正。
- 只能进行y和z方向上的位置校正。用户既无法实施方位校准，也无法在x方向（即路径方向 / 切线方向）上校准。

EGM的共同限制

- 只能在6轴机器人上使用EGM。
- 只能在涉及一台机器人的RAPID任务中使用EGM，即说无法在仅含附加轴的任务中使用EGM（也就是说在各个机器人目标点中，相关数据的pose部分存在着数值）。
- 必须在一个精确点上启动EGM移动。
- 每台机器人可只用一件外部装置来提供校正数据。

9.3.1.2 EGM Position Guidance介绍

什么是EGM Position Guidance

EGM Position Guidance是为高级用户设计的，它通过绕过相关路径规划（需要高响应性的机器人移动时可以采用这种规划）的方式为相关的机器人控制器提供了一个低级接口。用户可用EGM Position Guidance来高速读取相关运动系统的位置和向该系统高速写入位置（可达到每4毫秒一次，并伴随10到20毫秒的控制延迟。具体的延迟取决于相关的机器人类型）。可以用关节值或某种姿态来制定相关的引用项。用户可在EGM Position Guidance移动期间不会移动的任何工件上定义这种姿态。

EGM Position Guidance会处理一切必要的滤波、引用项监控和状态处理事宜。状态处理事宜包括程序启动 / 停止和紧急停止等。

与其它外部运动控制手段相比，EGM Position Guidance的主要优点在于较高的速率和较低的延时 / 等待时间。从“写入一个新位置”到“该给定位置开始影响实际的机器人位置”之间的时间通常约为20毫秒。

EGM会处理*Absolute Accuracy*。

EGM Position Guidance所不为之事

EGM会直接进入相关的电机引用项生成过程，即是说不会提供任何路径规划。这意味着您无法下令移到某个姿态目标点并期望这是一次直线移动。用户既无法下令进行一次指定速度下的移动，也无法下令进行一次应耗费指定时间的移动。

如果要下令进行此类需要路径规划的移动，那么您可使用RAPID中的标准移动指令，即MoveL和MoveJ等等。



警告

由于相关机器人控制器中的EGM绕过了路径规划，因此用户输入项会直接创建相应的机器人路径，所以很重要的一点就是确保发送给该控制器的位置引用项要尽量平顺。相关机器人会迅速对发送给相关控制器的所有位置引用项作出反应，连存在故障的机器人也是如此。

9.3.1.3 EGM Path Correction介绍

什么是EGM Path Correction

EGM Path Correction使用户或可校正一条编写好的机器人路径。用于测量实际路径的装置或传感器必须安装在相关机器人的工具法兰上，且该装置或传感器必须能够校准相应的传感器框架。

在相关的路径坐标系上实施校正。该坐标系的x轴来自相关路径的切线，其y轴为该路径切线的叉积，其z轴和已激活工具框架的z方向则是x轴和y轴的叉积。

EGM Path correction必须在精细点启动和结束。传感器测量结果可以在约48 ms的倍数时间提供。

9.3.2 使用EGM

9.3.2.1 基本方法

EGM Position Guidance的基本用法

如果用一件外部装置（传感器）来提供移动目标点，那么这就是移动 / 指引一台机器人的一般方式。

	操作
1	将相关机器人移到一个精确点处。
2	登记一个EGM客户端，然后获取一个EGM标识。之后要用该标识来把设置、激活、移动和停用与一种特定的EGM用法关联起来。EGM的状态仍为EGM_STATE_DISCONNECTED。
3	调用一条EGM设置指令来设置使用了信号或UdpUc协议连接的位置数据源。EGM的状态会变成EGM_STATE_CONNECTED。
4	选择是否将该位置作为关节值或一种姿态，并给出该位置的收敛标准（即何时视为已抵达该位置）。
5	如果选择了姿态，那么就定义使用哪个框架来定义相应的目标点位置以及在哪个框架中移动。
6	给出停止模式（一项可选的超时），然后执行移动本身。此时EGM的状态为EGM_STATE_RUNNING。这是相关机器人正在移动时的状态。
7	系统将在视为已抵达该位置（即是说已满足了相应的收敛标准）时停止EGM移动。此时EGM的状态又变回EGM_STATE_CONNECTED。

EGM Path Correction的基本用法

这是用EGM Path Correction校正一条编程路径的一般方式。

	操作
1	将相关机器人移到一个精确点处。
2	登记一个EGM客户端，然后获取一个EGM标识。之后要用该标识来把设置、激活、移动和停用与一种特定的EGM用法关联起来。EGM的状态仍为EGM_STATE_DISCONNECTED。
3	调用一条EGM设置指令来设置使用了信号或UdpUc协议连接的位置数据源。EGM的状态会变成EGM_STATE_CONNECTED。
4	定义相关传感器的校正框架（该框架始终属于工具框架）。
5	执行移动本身。此时EGM的状态为EGM_STATE_RUNNING。
	EGM将在下一个精确点处返回状态EGM_STATE_CONNECTED。
6	若要释放一个EGM标识以用于其它传感器，您就必须重置EGM，从而使EGM返回状态EGM_STATE_DISCONNECTED。

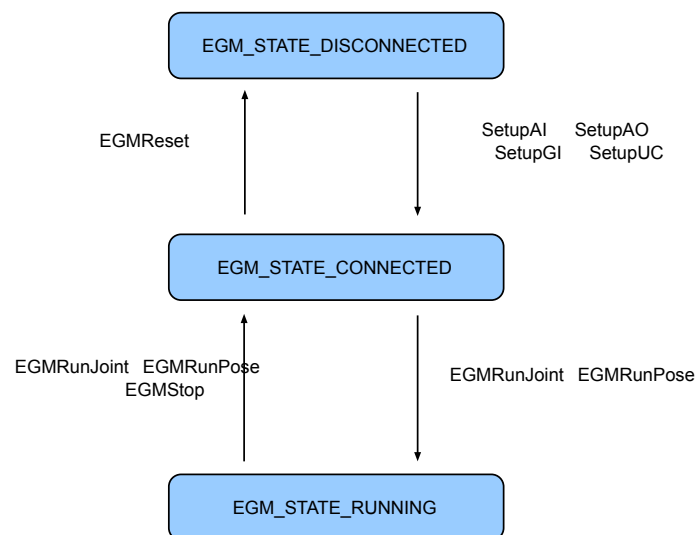
9.3.2.2 执行状态

描述

EGM进程拥有不同的状态：

值	描述
EGM_STATE_DISCONNECTED	未定义这一具体进程的EGM状态。 未激活任何设定。
EGM_STATE_CONNECTED	未激活指定的EGM进程。 已进行过设置，但并未激活任何EGM移动。
EGM_STATE_RUNNING	正在执行指定的EGM进程。 EGM移动处于激活状态，即是说移动了相关机器人。

不同状态之间的过渡情况如下图所示。



xx1400001082

RAPID指令EGMRunJoint和EGMRunPose始于EGM_STATE_CONNECTED，而只要还未达到相关目标点位置的收敛标准，或还未超过超时时间，那么这两条指令就会使相关状态变为EGM_STATE_RUNNING。当满足了其中一项条件时，EGM的状态会再次变为EGM_STATE_CONNECTED，而这两条指令则就此结束（即是说RAPID会继续执行下一条指令）。

如果EGM的状态为EGM_STATE_RUNNING，且停止了RAPID的执行过程，那么EGM则会进入状态EGM_STATE_CONNECTED。当程序重启时，EGM会返回状态EGM_STATE_RUNNING。

如果用从程序指针到主例程（PP to Main）或从PP到光标来移动相应的程序指针，那么EGM之前的状态又是EGM_STATE_RUNNING，那么EGM的状态就会变为EGM_STATE_CONNECTED。

9.3.2.3 输入数据

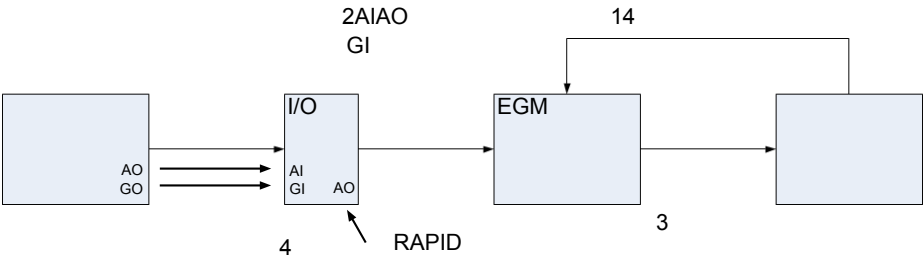
用于EGM Position Guidance的输入数据

用相关的EGM设置指令来选择输入数据来源。头三条指令会选择一个信号接口，而最后一条指令会选择一个UdpUc接口（*User Datagram Protocol Unicast Communication*）。

指令：	描述
EGMSetupAI	为EGM设置模拟输入信号
EGMSetupAO	为EGM设置模拟输出信号
EGMSetupGI	为EGM设置编组输入信号
EGMSetupUC	为EGM设置UdpUc协议

用于EGM的输入数据主要包含了作为关节或某种姿态的位置数据，也就是加上方位后的笛卡尔位置。

相关信号接口的数据流如下所示：

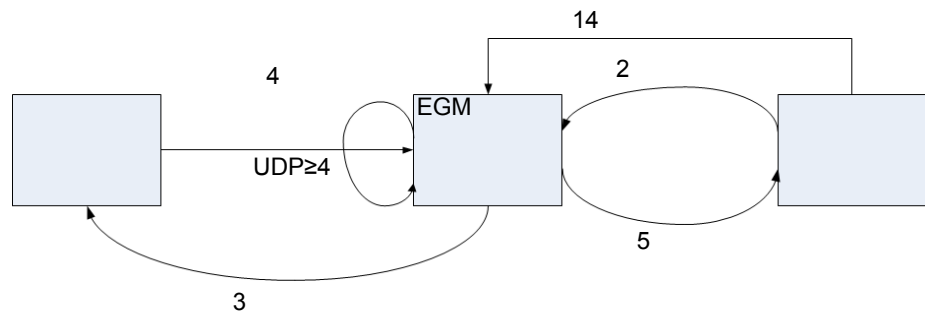


xx1400002016

- 1 运动控制会调用到EGM。
 - 2 EGM会读取相关信号的位置值。
 - 3 EGM会在运动控制中写入相关的位置数据。
- 相关传感器会把位置数据写入相应的信号中。

如果把信号作为数据源，那么该输入项就仅限于相关机器人的6，即6关节值或3笛卡尔位置值（x、y和z）再加上3欧拉角度值（rx、ry和rz），最高可达附加轴的6数值。

用于UdpUc接口的数据流如下所示：



xx1400002017

- 1 运动控制会调用到EGM。
 - 2 EGM会从运动控制中读取反馈数据。
 - 3 EGM会把反馈数据发送给相关的传感器。
 - 4 EGM会按相关传感器的消息来检查UDP队列。
 - 5 如果有一条消息，那么EGM会读取下一条消息，而到步骤5时则会在运动控制中写入相关的位置数据。如果没有发送任何位置数据，那么运动控制会继续使用EGM之前写入的最后一项位置数据。
- 传感器会向相应的控制器（EGM）发送位置数据。我们的建议是将这种发送与步骤3耦合起来，使该传感器与该控制器同相。

该控制环路是建立以下速度 - 位置关系上的：

$speed = k * (pos_ref - pos) + speed_ref$	k - 系数 pos_ref - 参考位置 pos - 所需位置 $speed_ref$ - 参考速度
---	--

至于如何用各种指令来执行针对某一外部装置的UdpUc协议，则请参见[第323页的EGM传感器协议](#)。该链接还描述了相关的输入数据。

用于EGM Path Correction的输入数据

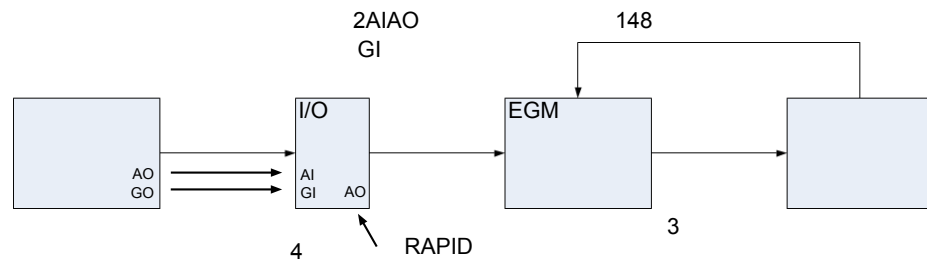
用相关的EGM设置指令来选择输入数据来源。头三条指令会选择一个信号接口，而最后一条指令会选择一个UdpUc接口（*User Datagram Protocol Unicast Communication*）。

指令：	描述
EGMSetupAI	为EGM设置模拟输入信号
EGMSetupAO	为EGM设置模拟输出信号
EGMSetupGI	为EGM设置编组输入信号
EGMSetupUC	为EGM设置UdpUc协议

用于EGM的输入数据主要包含了位置数据。

下一页继续

相关信号接口的数据流如下所示：



xx1400002016

- 1 运动控制会调用到EGM。
- 2 测量数据（y和x值）从信号读取或在48 ms的倍数时间从传感器获取。
- 3 EGM会计算相关的位置校正情况，然后将其写入运动控制中。如果采用了UdpUc协议，那么系统就会向相关传感器发送反馈。

9.3.2.4 输出数据

描述

输出数据仅适用于UdpUc接口。

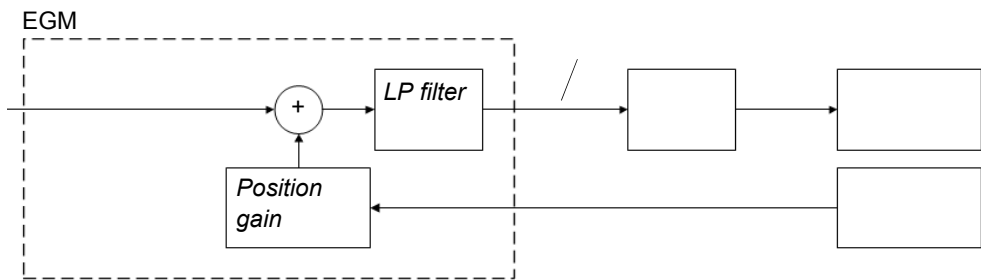
至于如何用各种指令来执行针对某一外部装置的UdpUc协议，则请参见[第323页的EGM传感器协议](#)。该链接还描述了相关的输出数据。

9.3.2.5 配置

用于EGM Position Guidance的配置

主题*Motion*下类型为*External Motion Interface Data*的系统参数会影响到EGM的行为。第327页的系统参数描述了所有可用的EGM参数。

下文更仔细地描述了会影响到EGM控制环路的两个参数。其中展示了EGM控制系统的一幅简图。



xx1400001083

<i>Default proportional Position Gain</i>	图中的参数位置增益会影响到前往目标点位置（由相应的传感器给定，并与当前的机器人位置有关）的响应移动。该数值越高，响应速度就越快。
<i>Default Low Pass Filter Bandwidth Time</i>	图中的参数LP Filter是过滤EGM的速度贡献量时所用的默认值。

9.3.2.6 框架

用于EGM Position Guidance的框架

EGM可在两种不同模式——关节模式和姿态模式——下运行，而下一节的内容仅适用于EGM姿态模式。

在关节模式下，由于传感器值和位置值都是相对于每根轴校准位置的给定轴角度（以度为单位），因此不需要参考框架。但姿态模式却离不开参考框架，毕竟系统只能以相对于参考框架的方式给出相关传感器的测量值和位置变动的方向。

RAPID指令`EGMActPose`定义了可在EGM中使用的所有框架：

框架	描述
工具	由可选 <code>\Tool</code> 自变数来定义将用于EGM进程的工具数据。
工件坐标	由可选 <code>\Wobj</code> 自变数来定义将用于EGM进程的工件数据。
校正	由强制 <code>CorrFrame</code> 自变数来定义给出最终移动方向时使用的框架。
传感器	由强制 <code>SensorFrame</code> 自变数来定义解读相关传感器数据时使用的框架。

工具与工件

只能按两种组合来定义这些工具和工件：

- 1 如果将工具连接到机器人上，那么就必须固定相应的工件。
- 2 如果固定住工具，那么就必须将相应的工件连接到机器人上。



注意

用户无法把工件或工具连接到非EGM机器人的任何其它机械单元上。

预定义框架类型

用户有必要了解框架`CorrFrame`和`SensorFrame`的关联情况。用数据类型`egmframetype`中的预定义框架类型来指定这一信息：

值	描述
<code>EGM_FRAME_BASE</code>	以相对于基本框架的方式来定义该框架（姿态模式）。
<code>EGM_FRAME_TOOL</code>	坐标系是相对于 <code>tool0</code> （姿势模式）定义的。
<code>EGM_FRAME_WOBJ</code>	坐标系是相对于当前工件（姿势模式）定义的。
<code>EGM_FRAME_WORLD</code>	以相对于全局框架的方式来定义该框架（姿态模式）。
<code>EGM_FRAME_JOINT</code>	这些数值为关节值（关节模式）。

用于EGM Path Correction的框架

EGM可在两种不同模式——关节模式和姿态模式——下运行，而下一节的内容仅适用于EGM姿态模式。

在关节模式下，由于传感器值和位置值都是相对于每根轴校准位置的给定轴角度（以度为单位），因此不需要参考框架。但姿态模式却离不开参考框架，毕竟系统只能以相对于参考框架的方式给出相关传感器的测量值和位置变动的方向。

下一页继续

RAPID指令EGMActPose定义了可在EGM中使用的所有框架：

框架	描述
工具	由可选\Tool自变数来定义将用于EGM进程的工具数据。
工件坐标	由可选\Wobj自变数来定义将用于EGM进程的工件数据。
校正	由强制CorrFrame自变数来定义给出最终移动方向时使用的框架。
传感器	由强制SensorFrame自变数来定义解读相关传感器数据时使用的框架。

工具与工件

只能按两种组合来定义这些工具和工件：

- 1 如果将工具连接到机器人上，那么就必须固定相应的工件。
- 2 如果固定住工具，那么就必须将相应的工件连接到机器人上。



注意

用户无法把工件或工具连接到非EGM机器人的任何其它机械单元上。

预定义框架类型

用户有必要了解框架CorrFrame和SensorFrame的关联情况。用数据类型egmframetype中的预定义框架类型来指定这一信息：

值	描述
EGM_FRAME_BASE	以相对于基本框架的方式来定义该框架（姿态模式）。
EGM_FRAME_TOOL	坐标系是相对于tool0（姿势模式）定义的。
EGM_FRAME_WOBJ	坐标系是相对于当前工件（姿势模式）定义的。
EGM_FRAME_WORLD	以相对于全局框架的方式来定义该框架（姿态模式）。
EGM_FRAME_JOINT	这些数值为关节值（关节模式）。

9.3.3 EGM传感器协议

描述

EGM传感器协议被设计用于以最少的经常性费用来实现机器人控制器与某个通信端点之间的高速通信。

通信端点通常为一个传感器，所以从此处起，我们将使用传感器来代替通信端点。有时该传感器会与一台PC相连，而该PC则会把传感器数据发送给相关机器人。这种传感器协议的用途是在相关的机器人控制器和传感器之间频繁交流传感器数据。EGM sensor protocol正在使用Google Protocol Buffers来进行编码，并把UDP则作为一则传送协议。选择Google Protocol Buffers的原因是其具有速度和语言中立性方面的优势。由于所发送的数据是高频发送的实时数据，且一旦丢失数据包，那么重新发送这些数据也无济于事，所以我们才选择了UDP作为传送协议。

由EGM proto文件定义EGM sensor protocol的数据结构。请在系统参数中配置相关的传感器名称、IP地址和传感器的端口号。最多能配置八个传感器。

该传感器正起到服务器的作用。在从相关机器人控制器处收到第一则消息前，该传感器无法向相关机器人发送任何内容；在收到第一则消息后，便可双向发送彼此独立的多则消息。采用了该协议的应用可能会对其使用情况作出一定限制，但该协议本身并不具备对请求响应或丢失消息监控的内置同步。这里没有专用的连接消息或断开消息，只有能双向独立流动的数据。来自相关机器人的第一则消息将是数据消息。此外用户还必须谨记，即使接收方的队列已满，UDP消息的发送方也仍会继续发送，所以接收方必须确保其队列是空的。

按照默认设定，不论传感器何时发送数据，相关机器人都会每隔4毫秒向该传感器发送一次数据或从该传感器读取一次数据。用户可用RAPID指令EGMActJoint或EGMActPose的可选自变数\SampleRate来把这一周期时间改为4毫秒的某个倍数。

Google Protocol Buffers

Google Protocol Buffers或*Protobuf*能非常高效地对数据进行串行化 / 去串行化。Protobuf大致上比XML快10到100倍。互联网上有大量关于Protobuf的信息，而从*Google overview*入手是个不错的选择。

简而言之，*.proto*文件描述了各种消息结构。此后要对*.proto*文件进行编译，而编译器则会生成供相关应用之后使用的串行化 / 去串行化代码。该应用会从相应的网络中读取一则消息，执行去串行化，创建一则消息，调用串行化方法，最后发送这一消息。

Protobuf具有语言中立性，所以大部分编程语言都或可使用Protobuf。不同的语言带来了许多不同的执行方式。

Protobuf的主要缺点是Protobuf消息会被串行化成某种二进制格式，这使用户更难通过网络分析仪来调试数据包。

第三方工具

除Google C++工具外，我们也验证了下列第三方工具和代码：

- *Nanopb*，生成C代码，且其无需分配任何动态内存。
- *Protobuf-net*，一个Google Protobuf .NET文库。
- *Protobuf-csharp*，一个Google Protobuf .NET文库，C# API类似于Google C++ API。

下一页继续



注意

注意上述代码为开源代码，这意味这您必须检查许可证，以确定是否允许在您的产品中使用相关代码。

EGM传感器协议说明

EGM传感器协议不是一项请求 / 响应协议，当传感器从相关机器人处获得第一则消息后，该传感器便可以任何频率发送数据。

EGM传感器协议有两种主要的数据结构，即*EgmRobot*和*EgmSensor*。其中*EgmRobot*由相关机器人发送，*EgmSensor*由I / O相关传感器发送。两种数据结构的所有消息字段都会被定义为可选，这意味着一则消息中可能有字段，也可能没有字段。如果某一应用使用了*Google Protocol Buffers*，则其必须检查是否存在可选字段。

*EgmHeader*由*EgmRobot*和*EgmSensor*共用。

```
message EgmHeader
{
  optional uint32 seqno = 1; // sequence number (to be able to find
    lost messages)
  optional uint32 tm = 2; // time stamp in milliseconds

  enum MessageType {
    MSGTYPE_UNDEFINED = 0;
    MSGTYPE_COMMAND = 1; // for future use
    MSGTYPE_DATA = 2; // sent by robot controller
    MSGTYPE_CORRECTION = 3; // sent by sensor
  }

  optional MessageType mtype = 3 [default = MSGTYPE_UNDEFINED];
}
```

变量	描述
seqno	序列号。 相关应用每发送一则消息，就应将序列号加一，这样才能检查一连串消息中是否有消息丢失。
tm	时间戳以毫秒为单位。 (可用于监测延时)。
mtype	消息类型。 相关传感器应设置成MSGTYPE_CORRECTION，相关机器人控制器应设置成MSGTYPE_DATA。

Google protobuf数据结构可包括重复元素（即一批类型相同的元素）。在EGM sensor protocol中，重复元素计数值最多为六个元素。

关于*EgmRobot*和*EgmSensor*的说明请参见*egm.proto*文件（[第339页的UdpUc代码示例](#)）。

如何用.Net构建一个EGM传感器通信端点

本指南假设您会用Visual Studio进行构建和编译，并熟悉其工作方式。

此处简单描述了如何用`protobuf-csharp-port`来安装和创建一项简单的测试应用。

	操作
1	从以下位置下载protobuf-csharp二元： https://code.google.com/p/protobuf-csharp-port/ 。
2	解压压缩文件。
3	将 <code>egm.proto</code> 文件复制到一个protobuf-csharp尚未解压的子目录中，比如 <code>~\protobuf-csharp\tools\egm</code> 。
4	在相关工具目录中启动一个Windows控制台，比如 <code>~\protobuf-csharp\tools</code> 。
5	在该Windows控制台输入以下内容，从而用 <code>egm.proto</code> 文件生成一份EGM C#文件（ <code>egm.cs</code> ）： <pre>protogen .\egm\egm.proto --proto_path=.\egm</pre>
6	在Visual Studio中创建一项C#控制台应用。 在Visual Studio中创建一项C# Windows控制台应用，比如 <code>EgmSensorApp</code> 。
7	安装NuGet，在Visual Studio中点击工具，然后点击扩展名管理器，转至在线，找到 <code>NuGet Package Manager extension</code> ，然后点击下载。
8	在使用NuGet的C# Windows Console应用的解决方案中安装protobuf-csharp。必须在Visual Studio中打开该解决方案。
9	在Visual Studio中依次选择工具、Nuget数据包管理器和数据包管理器控制台。 类型 <code>PM>Install-Package Google.ProtocolBuffers</code>
10	在Visual Studio项目中添加所生成的文件 <code>egm.cs</code> （添加现有项目）。
11	将示例代码复制到Visual Studio Windows Console应用文件（ <code>EgmSensorApp.cpp</code> ）中，然后依次编译、链接和执行。

如何用C++构建一个EGM传感器通信端点

用C++构建时无需其它第三方文库。

Google支持C++。在Windows中构建Google工具可能有点棘手，但这里会指导您如何构建针对Windows的protobuf。

当您已构建了`libprotobuf.lib`和`protoc.exe`时，则请使用以下程序：

	操作
1	执行Google protoc来生成访问等级 <code>protoc --cpp_out=. egm.proto</code>
2	创建一项win32控制台应用。
3	添加Protobuf源来作为包含目录。
4	在相关项目中添加所生成的 <code>egm.pb.cc</code> 文件，并从预编译标题中排除该文件。
5	复制 <code>egm-sensor.cpp</code> 文件的相关代码，具体请参见第339页的 <code>UdpUc</code> 代码示例。
6	编译并执行。

9 Engineering tools

9.3.3 EGM传感器协议

续前页

配置UdpUc装置

UdpUc最多通过Udp与八件装置进行通信。这些装置会起到服务器的作用，而相关的机器人控制器则会起到一个客户端的作用，即是说该机器人控制器会初始化相关传感器的连接状况。

系统参数

此处简述了配置一件装置时所用的相关参数。关于这些参数的更多信息请参见技术参考手册 - 系统参数。

这些参数属于主题*Communication*下的类型*Transmission Protocol*。

参数	描述
<i>Name</i>	相关传输协议的名称。 比如 <i>EGMsensor</i> 。
<i>Type</i>	传输协议的类型。 其必须是 <i>UDPUC</i> 。
<i>Serial Port</i>	将用于该传感器的串行端口的名称。 此处引用了类型为 <i>Serial Port</i> 的参数 <i>Name</i> 。 基于IP的传输协议（即 <i>Type</i> 具有数值TCP/IP、SOCKDEV、LTAPPTCP或UDPUC）不使用 <i>Serial Port</i> ，且具有数值N/A。
<i>Remote Address</i>	该远程装置的IP地址。
<i>Remote Port Number</i>	该远程装置已打开的IP端口号。

配置示例

若要让该装置为EGM提供输入数据，那么就必须按以下方式将其配置成一件UdpUc装置：

Name	Type	Serial Port	Remote Address	Remote Port Number
UCdevice	UDPUC	N/A	192.168.10.20	6510

做出这种配置改动后必须重启相关控制器，然后便可让EGM用该装置来指引一台机器人了。更多信息请参见第330页的使用带一件UdpUc装置的EGM *Position Guidance*。

9.3.4 系统参数

关于系统参数

此处简述了 *Externally Guided Motion* 所用的相关参数。关于这些参数的更多信息请参见技术参考手册 - 系统参数。

类型 *External Motion Interface Data*

Externally Guided Motion 所用的系统参数属于主题 *Motion* 下的类型 *External Motion Interface Data*。

参数	描述
<i>Name</i>	相关外部运动接口数据的名称。RAPID指令EGMSetupAI、EGMSetupAO、EGMSetupGI和EGMSetupUC中的参数 <i>ExtConfigName</i> 会引用该名称。
<i>Level</i>	外部运动接口登记决定了在哪个系统等级上实行校正。 0级与原物校正相对应，添加在紧靠伺服控制器的前侧。 1级会在校正时使用额外滤波，但这也会引入一些额外的延时和等待。 路径校正时必须使用2级。
<i>Do Not Restart After Motors Off</i>	决定了对紧急停止后的实例来说，是否宜在相关控制器处于“电机关闭”状态后自动重启外部运动接口。
<i>Return to Programmed Position when Stopped</i>	决定了当程序执行过程停止时，正在运行外部运动接口的各轴是否宜返回相应的编程位置。 如果为 <i>False</i> ，那么各轴将停在其当前位置处。 如果为 <i>True</i> ，那么各轴将移到相应的编程精确点处。
<i>Default Ramp Time</i>	定义了当停止外部运动接口的执行过程时，停止外部运动接口移动所需的默认总时间。 该数值将用于决定两项内容：当停止程序执行过程时，外部运动的速度贡献量宜以多快的速率减少至零；如果 <i>Return to Programmed Position when Stopped</i> 为 <i>True</i> ，那么各轴宜以多快的速率返回各自的编程位置。
<i>Default Proportional Position Gain</i>	定义了相关外部运动接口位置反馈控制的默认比例增益。更多信息请参见第320页的配置。
<i>Default Low Pass Filter Bandwidth</i>	定义了低通滤波器用来过滤外部运动接口执行过程的速度贡献量的默认带宽。更多信息请参见第320页的配置。

9.3.5 RAPID部件

关于RAPID组件

此处概述了*Externally Guided Motion*中的所有指令、函数和数据类型。

有关更多信息，请参阅 技术参考手册 - RAPID指令、函数和数据类型。

指令：

指令：	描述
EGMActJoint	EGMActJoint会激活一项特定的EGM进程，并为受传感器指引的关节移动定义所需的静态数据（即那些不会因EGM移动的不同而频繁改变的数据）。
EGMActMove	EGMActMove会激活一项特定的EGM进程，并为带路径校正的移动定义所需的静态数据（即不会因EGM路径校正移动的不同而频繁改变的数据）。
EGMActPose	EGMActPose会激活一项特定的EGM进程，并为受传感器指引的姿态移动定义所需的静态数据（即那些不会因EGM移动的不同而频繁改变的数据）。
EGMGetId	EGMGetId的作用是保留一个EGM标识（EGMid），之后便可在EGM的其它所有RAPID指令和函数中使用该标识，从而识别关联到所属RAPID运动任务上的某一EGM进程。 egmident的标识就是其名称，即是说如果用相同的egmident来第二次或第三次调用EGMGetId，那么系统既不会保留一项新的EGM进程，也不会更改其内容。
EGMMoveC	EGMMoveC的作用是沿圆弧将工具中心点（TCP）移到经过路径校正的给定目标点处。在进行这种移动时，相关姿态通常会相对于圆圈保持不变。
EGMMoveL	EGMMoveL的作用是沿直线将工具中心点（TCP）移到经过路径校正的给定目标点处。当该TCP保持固定时，用户也可用该指令来重定工具方位。
EGMReset	EGMReset重置了一项特定的EGM进程（EGMid），即是说取消保留。
EGMRunJoint	EGMRunJoint会从一项特定的EGM进程（EGMid）的一个精确点处执行一次受传感器指引的关节移动，并定义将要移动的关节。
EGMRunPose	EGMRunPose会从一项特定的EGM进程（EGMid）的一个精确点处执行一次受传感器指引的姿态移动，并定义将要改动的方向和方位
EGMSetupAI	EGMSetupAI的作用是为一项特定的EGM进程（EGMid）设置模拟输入信号，以作为指引相关机器人（最多外加6根附加轴）前往的位置目标点数值的来源。
EGMSetupAO	EGMSetupAO的作用是为一项特定的EGM进程（EGMid）设置模拟输出信号，以作为指引相关机器人（最多6根附加轴）前往的位置目标点数值的来源。
EGMSetupGI	EGMSetupGI的作用是为一项特定的EGM进程（EGMid）设置编组输入信号，以作为指引相关机器人（最多6根附加轴）前往的位置目标点数值的来源。
EGMSetupLTAPP	EGMSetupLTAPP的作用是为一项特定的EGM进程（EGMid）设置一项LTAPP协议，以作为路径校正的来源。
EGMSetupUC	EGMSetupUC的作用是为一项特定的EGM进程（EGMid）设置一件UdpUc装置，以作为指引相关机器人（最多6根附加轴）前往的位置目标点数值的来源。既可针对EGMRunJoint给出关节上的该位置，也可针对EGMRunPose给出笛卡尔格式的位置。
EGMStop	EGMStop会停止一项特定的EGM进程（EGMid）。

下一页继续

函数

函数	描述
EGMGetState	EGMGetState会检索一项EGM进程（EGMid）的状态。

数据类型

数据类型	描述
egmframetype	egmframetype的作用是为EGM中的校正和传感器测量定义所需的框架类型。
egmident	egmident会识别一项特定的EGM进程。
egm_minmax	egm_minmax的作用是定义结束EGM的收敛标准。
egmstate	egmstate的作用是为EGM中的校正和传感器测量定义所需的状态。
egmstopmode	egmstopmode的作用是为EGM中的校正和传感器测量定义所需的停止模式。

9.3.6 RAPID代码示例

9.3.6.1 使用带一件UdpUc装置的EGM Position Guidance

描述

若要让该装置为EGM提供输入数据，那么就必须首先将其配置成一件UdpUc装置。具体请参见第326页的配置UdpUc装置。

此时便可让EGM用该装置来指引一台机器人了。下面是一个简单的示例：

示例

```
MODULE EGM_test
VAR egmident egmID1;
VAR egmstate egmSt1;

! limits for cartesian convergence: +-1 mm
CONST egm_minmax egm_minmax_lin1:=[-1,1];
! limits for orientation convergence: +-2 degrees
CONST egm_minmax egm_minmax_rot1:=[-2,2];

! Start position
CONST jointtarget
  jpos10:=[[0,0,0,0,40,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
! Used tool
TASK PERS tooldata tFroniusCMT:=[TRUE,[[12.3313,-0.108707,416.142],
  [0.903899,-0.00320735,0.427666,0.00765917]],
  [2.6,[-111.1,24.6,386.6],[1,0,0,0],0,0,0.072]];
! corr-frame: wobj, sens-frame: wobj
TASK PERS wobjdata wobj_EGM1:=[FALSE,TRUE,"",
  [[150,1320,1140],[1,0,0,0]], [[0,0,0],[1,0,0,0]]];
! Correction frame offset: none
VAR pose corr_frame_offs:=[[0,0,0],[1,0,0,0]];

PROC main()
! Move to start position. Fine point is demanded.
MoveAbsJ jpos10\NoEOffs, v1000, fine, tFroniusCMT;
testuc;
ENDPROC

PROC testuc()
EGMReset egmID1;
EGMGetId egmID1;

egmSt1:=EGMGetState(egmID1);
TPWrite "EGM state: "\Num:=egmSt1;

IF egmSt1 <= EGM_STATE_CONNECTED THEN
! Set up the EGM data source: UdpUc server using device "EGMsensor:"
and
! configuration "default"
EGMSetupUC ROB_1, egmID1, "default", "EGMsensor:"\pose;
```

下一页继续

```

ENDIF

! Correction frame is the World coordinate system and the sensor
! measurements are relative
! to the tool frame of the used tool (tFroniusCMT)
EGMActPose egmID1\Tool:=tFroniusCMT, corr_frame_offs,
    EGM_FRAME_WORLD, tFroniusCMT.tframe, 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;

! Run: the convergence condition has to be fulfilled during 2
! seconds before RAPID
! executeion continues to the next instruction
EGMRunPose egmID1, EGM_STOP_HOLD \x \y \z \CondTime:=2
    \RampInTime:=0.05;

egmSt1:=EGMGetState(egmID1);
IF egmSt1 = EGM_STATE_CONNECTED THEN
TPWrite "Reset EGM instance egmID1";
EGMReset egmID1;
ENDIF
ENDPROC
ENDMODULE

```

9.3.6.2 使用带输入项信号的EGM Position Guidance

描述

必须在本系统的I / O配置中定义要和EGM搭配使用的所有信号，即那些用EGMSetupAI、EGMSetupAO或EGMSetupGI设置的信号。之后EGM便可用这些信号来指引一台机器人。

以下RAPID程序示例将模拟输出信号作为了输入项。采用模拟输出信号的主要原因是这些信号比模拟输入信号更容易仿真。在真实应用中，编组输入信号和模拟输入信号可能更为普遍。

为简便起见，我们将此例中的相关模拟输出信号设置成EGMRun指令前的一个恒定值。通常有一件外部装置来更新相关信号值，从而得出所需的机器人位置。

示例

```
MODULE EGM_test
VAR egmident egmID1;
VAR egmident egmID2;

CONST egm_minmax egm_minmax_lin1:=[-1,1];
CONST egm_minmax egm_minmax_rot1:=[-2,2];
CONST egm_minmax egm_minmax_joint1:=[-0.1,0.1];

CONST robtarget p20:=[[150,1320,1140],
  [0.000494947,0.662278,-0.749217,-0.00783173], [0,0,-1,0],
  [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget p30:=[[114.50,1005.42,1410.38],
  [0.322151,-0.601023,0.672381,0.287914], [0,0,-1,0],
  [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST jointtarget
  jpos10:=[[0,0,0,0,35,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST pose posecor:=[[1200,400,900],[1,0,0,0]];
CONST pose posesens:=[[12.3313,-0.108707,416.142],
  [0.903899,-0.00320735,0.427666,0.00765917]];

! corr-frame: world, sens-frame: world
VAR pose posecor0:=[[0,0,0],[1,0,0,0]];
VAR pose posesen0:=[[0,0,0],[1,0,0,0]];

TASK PERS tooldata tFroniusCMT:=[TRUE,[[12.3313,-0.108707,416.142],
  [0.903899,-0.00320735,0.427666,0.00765917]],
  [2.6,[-111.1,24.6,386.6],[1,0,0,0],0,0,0.072]];
TASK PERS loaddata load1:=[5,[0,1,0],[1,0,0,0],0,0,0];
! corr-frame: wobj, sens-frame: wobj
TASK PERS wobjdata
  wobj_EGM1:=[FALSE,TRUE,"",[[150,1320,1140],[1,0,0,0]],
  [[0,0,0],[1,0,0,0]]];
VAR pose posecor1:=[[0,0,0],[1,0,0,0]];
VAR pose posesen1:=[[0,0,0],[1,0,0,0]];
```

```

TASK PERS wobjdata
    wobj_EGM2:=[FALSE,TRUE,"",[[0,1000,1000],[1,0,0,0]],
    [[0,0,0],[1,0,0,0]]];
VAR pose posecor2:[[150,320,0],[1,0,0,0]];
VAR pose posesen2:[[150,320,0],[1,0,0,0]];

PROC main()
MoveAbsJ jpos10\NoEOffs, v1000, fine, tFroniusCMT;
testAO;
ENDPROC

PROC testAO()
! Get two different EGM identities. They will be used for two
different eGM setups.
EGMGetId egmID1;
EGMGetId egmID2;

! Set up the EGM data source: Analog output signals and
configuration "default"
! One guidance using Pose mode and one using Joint mode
EGMSetupAO ROB_1, egmID1, "default" \Pose \aoR1x:=ao_MoveX
\aoR2y:=ao_MoveY \aoR3z:=ao_MoveZ \aoR5ry:=ao_RotY
\aoR6rz:=ao_RotZ;
EGMSetupAO ROB_1, egmID2, "default" \Joint \aoR1x:=ao_MoveX
\aoR2y:=ao_MoveY \aoR3z:=ao_MoveZ \aoR4rx:=ao_RotX
\aoR5ry:=ao_RotY \aoR6rz:=ao_RotZ;

! Move to the starting point - fine point is needed.
MoveJ p30, v1000, fine, tool0;
! Set the signals
SetAO ao_MoveX, 150;
SetAO ao_MoveY, 1320;
SetAO ao_MoveZ, 900;
! Correction frame is the World coordinate system and the sensor
measurements are also relative to the world frame
! No offset is defined (posecor0 and posesen0)
EGMActPose egmID1 \Tool:=tFroniusCMT \WObj:=wobj0 \TLoad:=load1,
posecor0, EGM_FRAME_WORLD, posesen0, EGM_FRAME_WORLD
\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 \SampleRate:=16 \MaxPosDeviation:=1000;
! Run: keep the end position without returning to the start position
EGMRunPose egmID1,
EGM_STOP_HOLD\X\Y\Z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p20, v1000, fine, tFroniusCMT;
! Set the signals
SetAO ao_MoveX, 150;
SetAO ao_MoveY, 1320;
SetAO ao_MoveZ, 1100;

```

```
! Run with the same frame definitions: ramp down to the start
    position after having reached
! the EGM end position
EGMRunPose egmID1,
    EGM_STOP_RAMP_DOWN\x\y\z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p30, v1000, fine, tool0;
! Set the signals
SetAO ao_MoveX, 50;
SetAO ao_MoveY, -20;
SetAO ao_MoveZ, -20;
! Correction frame is the Work object wobj_EGM1 and the sensor
    measurements are also
! relative to the same work object. No offset is defined (posecor1
    and posesen1)
EGMActPose egmID1 \Tool:=tFroniusCMT \Wobj:=wobj_EGM1 \TLoad:=load1,
    posecor1, EGM_FRAME_WOBJ, posesen1, EGM_FRAME_WOBJ
    \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;
! Run: keep the end position without returning to the start position
EGMRunPose egmID1,
    EGM_STOP_HOLD\x\y\z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p20, v1000, fine, tFroniusCMT;
! Set the signals
SetAO ao_MoveX, 0;
SetAO ao_MoveY, 0;
SetAO ao_MoveZ, 0;
! Correction frame is the Work object wobj_EGM2 and the sensor
    measurements are also
! relative to the same work object. This time an offset is defined
    for the correction frame
! (posecor2), and for the sensor frame (posesen2)
EGMActPose egmID1 \Tool:=tFroniusCMT \Wobj:=wobj_EGM2 \TLoad:=load1,
    posecor2, EGM_FRAME_WOBJ, posesen2, EGM_FRAME_WOBJ
    \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;
! Run: keep the end position without returning to the start position
EGMRunPose egmID1,
    EGM_STOP_HOLD\x\y\z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p20, v1000, fine, tFroniusCMT;
! Set the signals
SetAO ao_MoveX, 0;
SetAO ao_MoveY, 0;
SetAO ao_MoveZ, 0;
```

```

! Correction frame is of tool type and the sensor measurements are
  relative to the work
! object wobj_EGM2. This time an offset is defined for the
  correction frame (posecor2), and
! for the sensor frame (posesen2)
EGMActPose egmID1 \Tool:=tFroniusCMT \WObj:=wobj_EGM2, posecor2,
  EGM_FRAME_TOOL, posesen2, EGM_FRAME_WOBJ \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\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p20, v1000, fine, tFroniusCMT\TLoad:=load1;
! Set the signals
SetAO ao_MoveX, 150;
SetAO ao_MoveY, 1320;
SetAO ao_MoveZ, 1100;
! Same as last, but with tool0 and wobj0
EGMActPose egmID1, posecor2, EGM_FRAME_TOOL, posesen2,
  EGM_FRAME_WOBJ \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;
! Run: keep the end position without returning to the start position
EGMRunPose egmID1,
  EGM_STOP_HOLD\x\y\z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p20, v1000, fine, tFroniusCMT\TLoad:=load1;
! Set the signals
SetAO ao_MoveX, 70;
SetAO ao_MoveY, -5;
SetAO ao_MoveZ, 0;
SetAO ao_RotX, 0;
SetAO ao_RotY, 0;
SetAO ao_RotZ, 0;
! Joint guidance for joints 2-6
EGMActJoint egmID2 \J2:=egm_minmax_joint1 \J3:=egm_minmax_joint1
  \J4:=egm_minmax_joint1 \J5:=egm_minmax_joint1
  \J6:=egm_minmax_joint1 \LpFilter:=20;
! Run: keep the end position without returning to the start position
EGMRunJoint egmID2, EGM_STOP_HOLD \J2 \J3 \J4 \J5 \J6 \CondTime:=0.1
  \RampInTime:=0.05 \PosCorrGain:=1;

EGMReset egmID1;
EGMReset egmID2;
ENDPROC
ENDMODULE

```

9.3.6.3 使用协议类型不同的EGM Path Correction

描述

此例包含了不同传感器和协议类型下的示例。其中所有传感器和协议类型都采用了相同的基本RAPID程序结构和相同的外部运动数据配置。

示例

```
MODULE EGM_PATHCORR
! Used tool
PERS tooldata tEGM:=[TRUE,[[148.62,0.25,326.31],
    [0.833900724,0,0.551914471,0]], [1,[0,0,100],
    [1,0,0,0],0,0,0]];
! Sensor tool, has to be calibrated
PERS tooldata
    tLaser:=[TRUE,[[148.619609537,50.250017146,326.310337954],
    [0.390261856,-0.58965743,-0.58965629,0.390263064]],
    [1,[-0.920483747,-0.000000536,-0.390780849],
    [1,0,0,0],0,0,0]];
! Displacement used
VAR pose PP:=[[0,-3,2],[1,0,0,0]];
VAR egmident egmId1;

! Protocol: LTAPP
! Example for a look ahead sensor, e.g. Laser Tracker
PROC Part_2_EGM_OT_Pth_1()
    EGMGetId egmId1;
    ! Set up the EGM data source: LTAPP server using device "Optsim",
    ! configuration "pathCorr", joint type 1 and look ahead sensor.
    EGMSetupLTAPP ROB_1, egmId1, "pathCorr", "OptSim", 1\LATR;
    ! Activate EGM and define the sensor frame.
    ! Correction frame is always the path frame.
    EGMActMove egmId1, tLaser.tframe\SampleRate:=50;
    ! Move to a suitable approach position.
    MoveJ p100,v1000,z10,tEGM\WObj:=wobj0;
    MoveL p110,v1000,z100,tEGM\WObj:=wobj0;
    MoveL p120,v1000,z100,tEGM\WObj:=wobj0;
    ! Activate displacement (not necessary but possible)
    PDispSet PP;
    ! Move to the start point. Fine point is demanded.
    MoveL p130, v10, fine, tEGM\WObj:=wobj0;
    ! movements with path corrections.
    EGMMoveL egmId1, p140, v10, z5, tEGM\WObj:=wobj0;
    EGMMoveL egmId1, p150, v10, z5, tEGM\WObj:=wobj0;
    EGMMoveC egmId1, p160, p165, v10, z5, tEGM\WObj:=wobj0;
    ! Last path correction movement has to end with a fine point.
    EGMMoveL egmId1, p170, v10, fine, tEGM\WObj:=wobj0;
    ! Move to a safe position after path correction.
    MoveL p180,v1000,z10,tEGM\WObj:=wobj0;
    ! Release the EGM identity for reuse.
    EGMReset egmId1;
```


ENDPROC

```

! Protocol: LTAPP
! Example for an at point sensor, e.g. Weldguide
PROC Part_2_EGM_WG_Pth_1()
  EGMGetId egmId1;
  ! Set up the EGM data source: LTAPP server using device "wglsim",
  ! configuration "pathCorr", joint type 1 and at point sensor.
  EGMSetupLTAPP ROB_1, egmId1, "pathCorr", "wglsim", 1\APTR;
  ! Activate EGM and define the sensor frame,
  ! which is the tool frame for at point trackers.
  ! Correction frame is always the path frame.
  EGMActMove egmId1, tEGM.tframe\SampleRate:=50;
  ! Move to a suitable approach position.
  MoveJ p100,v1000,z10,tEGM\WObj:=wobj0;
  MoveL p110,v1000,z100,tEGM\WObj:=wobj0;
  MoveL p120,v1000,fine,tEGM\WObj:=wobj0;
  ! Activate displacement (not necessary but possible)
  PDispSet PP;
  ! Move to the start point. Fine point is demanded.
  MoveL p130, v10, fine, tEGM\WObj:=wobj0;
  ! movements with path corrections.
  EGMMoveL egmId1, p140, v10, z5, tEGM\WObj:=wobj0;
  EGMMoveL egmId1, p150, v10, z5, tEGM\WObj:=wobj0;
  EGMMoveC egmId1, p160, p165, v10, z5, tEGM\WObj:=wobj0;
  ! Last path correction movement has to end with a fine point.
  EGMMoveL egmId1, p170, v10, fine, tEGM\WObj:=wobj0;
  ! Move to a safe position after path correction.
  MoveL p180,v1000,z10,tEGM\WObj:=wobj0;
  ! Release the EGM identity for reuse.
  EGMReset egmId1;
ENDPROC

```

```

! Protocol: UdpUc
! Example for an at point sensor, e.g. Weldguide
PROC Part_2_EGM_UDPUC_Pth_1()
  EGMGetId egmId1;
  EGMSetupUC ROB_1, egmId1, "pathCorr", "UCdevice"\PathCorr\APTR;
  EGMActMove egmId1, tEGM.tframe\SampleRate:=50;
  ! Move to a suitable approach position.
  MoveJ p100,v1000,z10,tEGM\WObj:=wobj0;
  MoveL p110,v1000,z100,tEGM\WObj:=wobj0;
  MoveL p120,v1000,fine,tEGM\WObj:=wobj0;
  ! Activate displacement (not necessary but possible)
  PDispSet PP;
  ! Move to the start point. Fine point is demanded.
  MoveL p130, v10, fine, tEGM\WObj:=wobj0;
  ! movements with path corrections.
  EGMMoveL egmId1, p140, v10, z5, tEGM\WObj:=wobj0;
  EGMMoveL egmId1, p150, v10, z5, tEGM\WObj:=wobj0;
  EGMMoveC egmId1, p160, p165, v10, z5, tEGM\WObj:=wobj0;

```

下一页继续

9.3.6.3 使用协议类型不同的EGM Path Correction

续前页

```
! Last path correction movement has to end with a fine point.
EGMMoveL egmId1, p170, v10, fine, tEGM\WObj:=wobj0;
! Move to a safe position after path correction.
MoveL p180,v1000,z10,tEGM\WObj:=wobj0;
! Release the EGM identity for reuse.
EGMReset egmId1;
ENDPROC
ENDMODULE
```

9.3.7 UdpUc代码示例

文件位置

本RobotWare版本有以下代码示例可用。

文件	描述
<i>egm-sensor.cs</i>	使用了protobuf-csharp-port的示例
<i>egm-sensor.cpp</i>	使用了Google protocol buffers C++的示例
<i>egm.proto</i>	<i>egm.proto</i> 定义了相关机器人与相关传感器之间的数据契约。

可从相应的PC或IRC5控制器处获得这些文件。

- 在RobotStudio的RobotWare安装文件夹中：...\\RobotPackages\\RobotWare_RPK_<version>\\utility\\Template\\EGM\\
- 在相应的IRC5控制器上：
<SystemName>\\PRODUCTS\\<RobotWare_xx.xx.xxxx>\\utility\\Template\\EGM\\



注意

右键单击插件浏览器中的已安装RobotWare版本，然后选择打开数据包文件夹，系统便会从RobotStudio插件标签处将您引导至RobotWare安装文件夹。

9.4 Robot Reference Interface 【包括在689-1中】

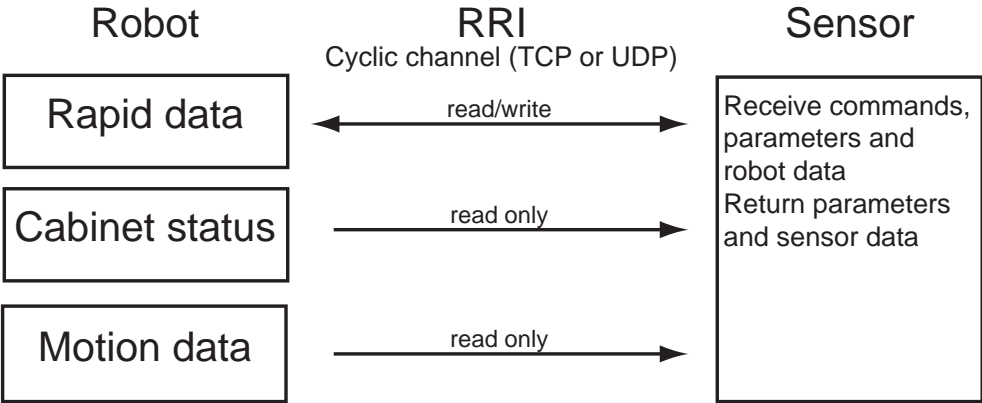
9.4.1 Robot Reference Interface介绍

简介

RobotWare选项*Externally Guided Motion*中包括了*Robot Reference Interface*。
*Robot Reference Interface*支持周期通道上的数据交换，使系统或可一方面定期发送相关机器人控制器的计划与实际机器人位置数据函数，另一方面交换来自或送往相关机器人控制器的其它RAPID变量。系统会用XML格式来表达相应的消息内容，同时用合适的传感器配置文件来配置这些内容。

Robot Reference Interface

用户可在IRC5控制器的高优先级网络环境（可以此确保高达250赫兹的稳定数据交换）下运行周期通信通道（TCP或UDP）。



xx0800000128

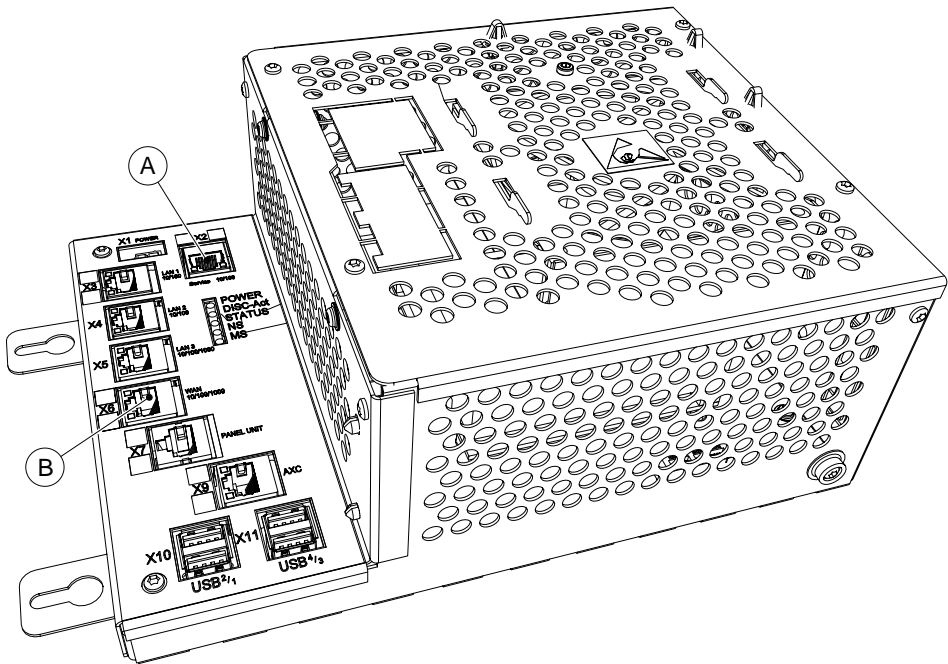
9.4.2 安装

9.4.2.1 连接通信电缆


概述

本节描述了在控制器的何处连接相应的通信电缆。更多使用说明则请参见您机器人系统的相应产品手册。

Location



xx1300000609

A	相关计算机单元上的服务端口（与相关控制器上的服务端口相连）	
B	相关计算机单元上的WAN端口	
	操作	注释
1	使用这两种连接之一（A或B）。	<div> 注意</div> <div>只有当该服务连接处于闲置状态时，才能使用该服务连接。</div>

9.4.2.2 先决条件

概述

本节描述了使用 *Robot Reference Interface* 的先决条件。

UDP/IP或TCP IP

Robot Reference Interface 支持通过标准IP协议UDP或TCP进行通信。

建议

总体通信中的延时主要取决于所用网络的拓扑结构。在交换网络中，缓冲各交换机中的消息会导致传输延时；而在并行网络中，与多个通信合作伙伴间的冲突则会导致系统重新发送消息。

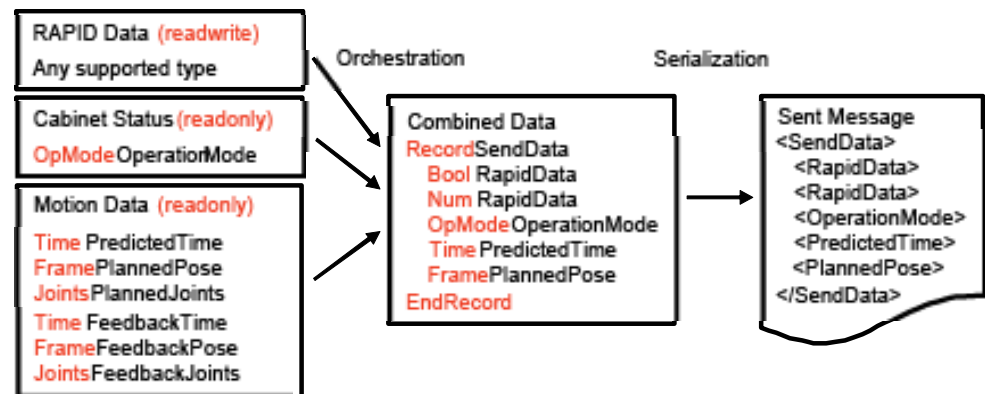
所以我们建议用外部系统与相关机器人控制器之间的专用以太网链路来实现实时应用中的必要操作。用户既可用 *Robot Reference Interface* 来与任何基于处理器并支持以太网IP的装置进行通信，也可用该接口将数据串行成XML格式。

9.4.2.3 数据编配

概述

用户可将相关RAPID等级上任何数据的发文消息与机柜和运动主题的内部数据结合起来。若要在相关装置配置中定义数据编配，就把内部链接数据的“链路”属性设置成 *Intern*。

图示



xx0800000178

Controller主题的数据

名称	类型	描述	备注
OperationMode	OpMode	相关机器人的运行模式。	可在相关配置文件中定义OpMode类型下各元的映射状况。

Motion主题的数据

名称	类型	描述	备注
FeedbackTime	时间	驱动器反馈的机器人位置时间戳。	有大约8毫秒的延时。
FeedbackPose	框架	用驱动器反馈计算出的机器人TCP。	系统会用当前的工具和工件进行计算。
FeedbackJoints	关节	从驱动器反馈中收集的机器人关节值。	
PredictedTime	时间	所计划机器人TCP位置的时间戳与关节值。	预测时间约为24毫秒到60毫秒，具体取决于机器人的类型。
PlannedPose	框架	所计划机器人TCP。	系统会用当前的工具和工件进行计算。
PlannedJoints	关节	所计划机器人关节值。	

9.4.2.4 受支持的数据类型

概述

本节简短描述了*Robot Reference Interface*支持的数据类型。至于受支持数据类型方面的详细信息，则请参见[第13页的参考信息](#)。

数据类型

*Robot Reference Interface*支持下列简单数据类型：

数据类型	描述	RAPID类型映射
bool	布尔（Boolean）值。	bool
real	单精度，浮点值。	num
time	以浮点值表达的时间（以秒为单位）。	num
string	最多80个字符的字符串。	string
frame	笛卡尔位置和以欧拉（Euler）角为单位的方位（横滚 - 俯仰 - 偏转）。	pose
joint	机器人关节值。	robjoint

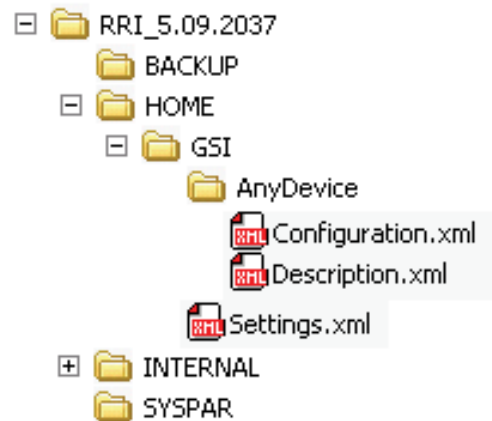
此外还能将用户定义记录（由受支持的简单数据类型构成）从外部系统转移到相应的机器人控制器上。用户必需在外部装置的配置文件中指定用户定义记录的类型。至于如何创建用户定义记录类型，则请参见[第349页的装置配置](#)。

9.4.3 配置

9.4.3.1 接口配置

配置文件

该接口的配置和设定文件必须位于文件夹HOME/GSI中，从而确保将这些配置文件纳入系统备份。



xx0800000177

相关信息

*Settings.xml*文件的更多详细信息请参见[第346页的接口设定](#)。

*Description.xml*文件的更多详细信息请参见[第347页的装置说明](#)。

*Configuration.xml*文件的更多详细信息请参见[第349页的装置配置](#)。

9.4.3.2 接口设定

概述

本节描述了xml文件*Settings.xml*的用法。

Settings.xml

设定文件*Settings.xml*包含了GSI接口的一般设定。该文件位于文件夹HOME/GSI中。对选项*Robot Reference Interface*而言，该文件会为安装在相关控制器上的外部系统引用所有的通信客户端。用户可按XML图解 *Settings.xsd*来定义该*Settings.xml*文件。

示例

对安装在相关控制器上的每个通信客户端而言，文件*Settings.xml*必须包含*Clients*一节中的一个*Client*条目。对仅支持CDP的*Robot Reference Interface*选项而言，*Convention*属性为识别相关客户端采用的协议规约。*Name*属性会识别相关客户端的名称，并指出含有装置相关配置文件的文件夹。

```
<?xml version="1.0" encoding="UTF-8"?>
<Settings>
  <Clients>
    <Client Convention="CDP" Name="MySensor" />
  </Clients>
</Settings>
```

CDP意指周期性数据协议，同时也是传输 *Robot Reference Interface*消息时该协议的内部名称。

系统将为该接口模块创建一个内部客户节点，而该节点则能与执行数据服务器应用的外部系统*MySensor*相连，并通过*Robot Reference Interface*与相关机器人进行通信。对每套传感器系统而言，用该传感器标识符（比如*MySensor*）命名的子目录中包含了更多设定。

9.4.3.3 装置说明

概述

本节描述了xml文件*Description.xml*的用法。

Description.xml

装置说明文件Description.xml位于该装置的相应子目录中，其指定了某个已安装装置的一般装置参数、网络连接状况和CDP特有的通信设定。用户可按XML图解Description.xsd来定义装置说明。

示例

以下是一项装置说明示例：

```
<?xml version="1.0" encoding="utf-8"?>
<Description>
  <Name>AnyDevice</Name>
  <Convention>CDP</Convention>
  <Type>IntelligentCamera</Type>
  <Class>MachineVision</Class>
  <Network Address="10.49.65.74" Port="Service">
    <Channel Type="Cyclic" Protocol="Udp" Port="3002" />
  </Network>
  <Settings>
    <TimeOut>2000</TimeOut>
    <MaxLost>30</MaxLost>
    <DryRun>false</DryRun>
  </Settings>
</Description>
```

Name

第一节定义了相关的一般装置参数。Name元件会识别所述装置的名称，并宜与设定文件中指定的装置名称相对应。由于系统一开始会用相关描述符的名称来引用相应RAPID指令中的该装置，因此其必须对应着在RAPID等级上为该装置描述符指定的标识符。

组件	属性	描述	值	备注
Name		装置标识符	任何字符串	最多16个字符

Convention

对仅支持Cyclic Data Protocol (CDP) 的 *Robot Reference Interface*选项而言，Convention元件会识别相关装置宜采用的协议。

组件	属性	描述	值	备注
Convention		协议类型	CDP	

Type和Class

Type和Class元件会识别相应的装置类型和等级，且这些元件此时并未生效，所以能包含尚未定义的装置类型或等级。

组件	属性	描述	值	备注
Type		传感器类型	任何字符串	未生效

下一页继续

9 Engineering tools

9.4.3.3 装置说明

续前页

组件	属性	描述	值	备注
Class		传感器等级	任何字符串	未生效

Network

Network一节定义了该装置的网络连接设定。Address属性指定了该装置在相关网络上的IP地址或主机名称。可选的Port属性的作用是指定控制器一侧插入电缆用的物理以太网端口。有效值为WAN和Service。若将WAN用于通信，则可忽略该属性。

组件	属性	描述	值	备注
Network		网络设定		
	Address	该装置的IP地址或主机名称	任何有效的IP地址或主机名称	10.49.65.249 DE-L-0328122
	Port	控制器上的物理以太网端口	WAN Service	可选。如果使用了WAN端口，那么可以忽略。

Channel

Channel元件会识别相关机器人控制器与外部装置之间的通信设定，而Type属性则定义了相应的通道类型（*Robot Reference Interface*仅支持Cyclic）。

Protocol属性会识别相关通道上使用的IP协议，而对于*Robot Reference Interface*，您则可指定是使用Tcp还是Udp。Port属性会为相关装置一侧的通道指定相应的逻辑端口号。

组件	属性	描述	值	备注
Channel		通道设定		
	Type	通道类型	Cyclic	
	Protocol	IP协议类型	Tcp Udp	
	Port	相关通道的逻辑端口号	无符号短整型 (uShort)	该装置上任何可用的端口号，最大为65535。

Settings

Settings一节包含了CDP协议特有的通信参数。TimeOut元素定义了不接收消息的超时时间。该元素会识别这一时间，直至认为连接中断为止。只有双向通信时才需要该元素。MaxLost属性定义了最多允许不接受或丢失多少则消息。DryRun元素会识别是否监控了消息接受情况，以及是否能用这种接受情况来设置单向通信。

组件	描述	值	备注
TimeOut	通信超时		时间（以毫秒为单位，且为4毫秒的倍数）。
MaxLost	文件包的最大允许损失	整数	
DryRun	接口执行模式	弯曲件	如果为TRUE，那么就不会检查TimeOut和MaxLost。

如果Description.xml中的元素DryRun被设置成FALSE，那么系统就会采用超时和最大丢失的设定，在*Robot Reference Interface*上建立通信监控。这种监控需要所关联的装置来回答相关机器人控制器发出的的每一则消息，而如果超过了最长响应时间或丢失包的最大数目，那么这种监控就会生成一项通信错误。发出的的每一则消息都有一个ID，而相应的回复中也需要用该ID来识别回复消息和检测丢失了哪些文件包。也可参见[第355页的发送XML消息](#)中的示例。

9.4.3.4 装置配置

概述

装置配置文件`Configuration.xml`位于相关装置的对应子目录中，其定义了该装置采用的枚举类型和复杂类型，并会识别相应的可用参数（可为周期传输预定这些参数）。用户可按XML图解`Configuration.xsd`来定义该配置文件。以下文件展示了一种简化后的装置配置。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <Enums>
    <Enum Name="opmode" Link="Intern">
      <Member Name="ReducedSpeed" Alias="Alias"/>
    </Enum>
  </Enums>
  <Records>
    <Record Name="senddata">
      <Field Name="PlannedPose" Type="Pose" Link="Intern" />
    </Record>
  </Records>
  <Properties>
    <Property Name="DataToSend" Type="senddata" Flag="WriteOnly" />
  </Properties>
</Configuration>
```

Enums

Enums节中的每个Enum都定义了一种枚举类型。Enum元素的Name属性指定了相应枚举类型的名称，而可选的Link属性则会识别该枚举类型下的各成员间是否存在内部链接。

组件	属性	描述	值	备注
Enum	Name	枚举类型的名称	一个有效的RAPID符号名称	最多16个字符。
	Link	枚举类型下各成员间的链接	<i>Intern</i>	可选。如果各成员仅有RAPID链接，那么可以忽略。

Member

每个Member元素都定义了相关枚举类型下的一个成员元素。Name属性指定了控制器一侧的相关成员名称（在RAPID等级上）。Alias属性会识别装置一侧（以及所发送消息中）的相关成员名称。

组件	属性	描述	值	备注
Member	Name	枚举类型成员的名称	一个有效的RAPID符号名称	最多16个字符。有效的内部RAPID符号名称，具体请参见 第343页的数据编配 。
	Alias	枚举类型成员的别名	字符串	可选。该别名用于相关装置一侧和消息中

下一页继续

9 Engineering tools

9.4.3.4 装置配置

续前页

Record

Records节中的每个Record元素都定义了一种复杂类型的一项声明。RAPID将把这种复杂类型表示为一项RECORD声明。Name属性会识别控制器一侧的相关复杂类型的名称，而Alias属性则定义了装置一侧和消息中的该类型别名。

组件	属性	描述	值	备注
Record	Name	该复杂类型的名称。	一个有效的RAPID符号名称	最多16个字符。
	Alias	复杂类型的别名。	字符串	可选。该别名用于相关装置一侧和消息中。

Field

每个Field元素都定义了一种复杂类型下的一个字段元素。Name属性会识别该字段的名称，Type属性会识别与该字段有关的枚举类型、复杂类型或简单类型，Size属性定义了一个多维字段的大小，而Link属性则会识别该字段是否存在内部链接。

组件	属性	描述	值	备注
Field	Name	该复杂类型字段的名称	一个有效的RAPID符号名称	最多16个字符。有效的内部RAPID符号名称，具体请参见 第343页的数据编配 。
	Type	该字段的数据类型 中断的数据类型	所有受支持的数据类型	详情请参阅 第344页的受支持的数据类型 一节。
	Size	该字段的维度（数组大小）	整数	可选。只有基本类型才能被定义为数组。
	Link	复杂类型字段的链接	<i>Intern</i>	可选。如果字段拥有RAPID链接，那么可以忽略。
	Alias	复杂类型字段的别名	字符串	可选。该别名用于装置一侧和消息中。

Properties

Properties节中的每个Property元素都定义了一个可用于SiGetCyclic和SiSetCyclic指令的RAPID变量。

组件	属性	描述	值	备注
Property	Name	该性质的名称	一个有效的RAPID符号名称	最多16个字符。
	Type	该性质的数据类型	所有受支持的数据类型	详情请参阅 第344页的受支持的数据类型 一节。
	Size	维度（数组大小）	整数	可选。只有基本类型才能被定义为数组。
	Flag	访问旗标	无 <i>ReadOnly</i> <i>WriteOnly</i> <i>ReadWrite</i>	可选。如果启用了性质读写，那么可以忽略。
	Link	性质的链接	<i>Intern</i>	如果字段拥有RAPID链接，那么则具有强制性。
	Alias	该性质的别名	字符串	可选。该别名用于装置一侧和消息中。

9.4.4 配置示例

9.4.4.1 RAPID编程

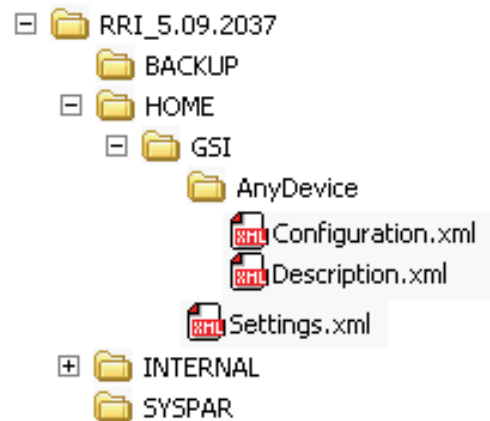
RAPID模块

必须创建并载入一个包含对应RAPID记录声明和变量声明的RAPID模块。
RobotWare中不包括FlexPendant示教器用户界面。

9.4.4.2 配置示例

概述

文件Settings.xml、Description.xml和Configuration.xml位于文件夹HOME\GSI\中



xx0800000177



注意

该文件夹的名称必须对应着相关装置的名称。具体请参见第347页的装置说明。我们已在此例中使用了名称AnyDevice。

Description.xml中采用的网络地址是指运行服务器的PC，而不是相关的机器人控制器。具体请参见第347页的装置说明。

Settings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Settings>
  <Servers>
  <Servers/>
  <Clients>
    <Client Convention="CDP" Name="AnyDevice" />
  </Clients>
</Settings>
```

Description.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Description>
  <Name>AnyDevice</Name>
  <Convention>CDP</Convention>
  <Type>IntelligentCamera</Type>
  <Class>MachineVision</Class>
  <Network Address="10.49.65.74" Port="Service">
    <Channel Type="Cyclic" Protocol="Udp" Port="3002" />
  </Network>
  <Settings>
    <TimeOut>2000</TimeOut>
    <MaxLost>30</MaxLost>
  </Settings>
</Description>
```

下一页继续


```

    <DryRun>false</DryRun>
  </Settings>
</Description>

```

Configuration.xml

```

<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
  <Enums>
    <Enum Name="OperationMode" Link="Intern">
      <Member Name="Automatic" Alias="Auto" />
      <Member Name="ReducedSpeed" Alias="ManRS" />
      <Member Name="FullSpeed" Alias="ManFS" />
    </Enum>
  </Enums>
  <Records>
    <Record Name="RobotData">
      <Field Name="OperationMode" Type="OperationMode" Link="Intern"
        Alias="RobMode" />
      <Field Name="FeedbackTime" Type="Time" Link="Intern"
        Alias="Ts_act" />
      <Field Name="FeedbackPose" Type="Frame" Link="Intern"
        Alias="P_act" />
      <Field Name="FeedbackJoints" Type="Joints" Link="Intern"
        Alias="J_act" />
      <Field Name="PredictedTime" Type="Time" Link="Intern"
        Alias="Ts_des" />
      <Field Name="PlannedPose" Type="Frame" Link="Intern"
        Alias="P_des" />
      <Field Name="PlannedJoints" Type="Joints" Link="Intern"
        Alias="J_des" />
      <Field Name="ApplicationData" Type="Num" Size="18"
        Alias="AppData" />
    </Record>
    <Record Name="SensorData">
      <Field Name="ErrorString" Type="String" Alias="EStr" />
      <Field Name="ApplicationData" Type="Num" Size="18"
        Alias="AppData" />
    </Record>
  </Records>
  <Properties>
    <Property Name="RobData" Type="RobotData" Flag="WriteOnly"/>
    <Property Name="SensData" Type="SensorData" Flag="ReadOnly"/>
  </Properties>
</Configuration>

```

RAPID配置

这是一个RRI执行示例。其中“出”数据使用了一个18数的数组（rob数据），而“入”数据则收到了一段字符串和一个18数的数组（sens数据）。需要按照文件 configuration.xml来进行定义。

```

RECORD applicationdata
  num Item1;

```

下一页继续

```
num Item2;
num Item3;
num Item4;
num Item5;
num Item6;
num Item7;
num Item8;
num Item9;
num Item10;
num Item11;
num Item12;
num Item13;
num Item14;
num Item15;
num Item16;
num Item17;
num Item18;
ENDRECORD
RECORD robdata
  applicationdata AppData;
ENDRECORD
RECORD sensdata
  string ErrString;applicationdata AppData;
ENDRECORD
! Sensor Declarations
PERS sensor AnyDevice := [1,4,0];
PERS robdata DataOut := [[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]];
! Setup Interface Procedure
PROC RRI_Open()
  SiConnect AnyDevice;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, 64;
  SiSetCyclic AnyDevice, DataOut, 64;
ENDPROC
! Close Interface Procedure
PROC RRI_Close()
  ! Close the connection
  SiClose RsMaster;
ENDPROC
ENDMODULE
```

发送XML消息

每则XML消息都有一个以根元素为名、并具有属性Id（消息ID）和Ts（该消息的时间戳）的数据变量。此后记录字段便是相应的子元素。系统会用属性来表达一个多值字段（数组或记录）的数值。

机器人控制器发出的消息

相应的时间单位为秒（浮动），分辨率为1毫秒。相应的位置（长度）单位为毫米（浮动）。相应的位置（角度）单位为弧度。

名称	数据类型	描述
Id	整数	最后接受的机器人数据消息ID
Ts	浮动	时间戳（消息）
RobMode	运行模式	运行模式
TS_act	浮动	时间戳（实际位置）
P_act	姿态	实际笛卡尔位置
J_act	接点	实际关节位置
TS_des	浮动	时间戳（所需位置）
P_des	姿态	所需笛卡尔位置
J_des	接点	所需关节位置
AppData	18个浮点的数组	自由定义的应用数据

```
<RobData Id="111" Ts="1.202" >
  <RobMode>Auto</RobMode>
  <Ts_act>1.200</Ts_act>
  <P_act X="1620.0" Y="1620.0" Z="1620.0" Rx="100.0" Ry="100.0"
    Rz="100.0" />
  <J_act J1="1.0" J2="1.0" J3="1.0" J4="1.0" J5="1.0" J6="1.0" />
  <Ts_des>1.200</Ts_des>
  <P_des X="1620.0" Y="1620.0" Z="1620.0" Rx="100.0" Ry="100.0"
    Rz="100.0" />
  <J_des J1="1.0" J2="1.0" J3="1.0" J4="1.0" J5="1.0" J6="1.0" />
  <AppData X1="1" X2="1620.000" X3="1620.000" X4="1620.000"
    X5="1620.000" X6="1620.000" X7="1620.000" X8="1620.000"
    X9="1620.000" X10="1620.000" X11="1620.000" X12="1620.000"
    X13="1620.000" X14="1620.000" X15="1620.000" X16="1620.000"
    X17="1620.000" X18="1620.000" />
</RobData>
```

从机器人控制器收到的消息

时间单位为秒（浮点）。

名称	数据类型	描述
Id	整数	最后收到的数据消息的ID。该ID必须对应着相关机器人控制器发出的ID。
Ts	浮动	时间戳
EStr	字符串	错误消息
AppData	18个浮点的数组	自由定义的应用数据

网络上的对应XML消息将表现为以下形式：

```
<SensData Id="111" Ts="1.234">
  <EStr>xxxx</Estr>
  <AppData X1="232.661" X2="1620.293" X3="463.932"
    X4="1231.053" X5="735.874" X6="948.263" X7="2103.584"
    X8="574.228" X9="65.406" X10="2372.633" X11="20.475"
    X12="96.729" X13="884.382" X14="927.954" X15="748.294"
    X16="3285.574" X17="583.293" X18="684.338" />
</SensData>
```

9.4.5 RAPID组件

关于RAPID部件

此处概述了*Robot Reference Interface*中的所有指令、函数和数据类型。
有关更多信息，请参阅 技术参考手册 - *RAPID*指令、函数和数据类型。

指令：

指令：	描述
SiConnect	传感器接口连接
SiClose	传感器接口关闭
SiGetCyclic	传感器接口获取周期
SiSetCyclic	传感器接口设置周期

函数

*Robot Reference Interface*中不包括任何函数。

数据类型

数据类型	描述
sensor	外部装置描述符
sensorstate	该装置的通信状态

此页刻意留白

10 Tool control options

10.1 Servo Tool Change [630-1]

10.1.1 概述

目的

“伺服工具更换”能够在线更换各种工具。

有了选项“伺服工具更换”，用户就能断开通往某附加轴电机的电缆，然后将该电缆与另一根附加轴的电机相连。可以在生产运行中进行这一操作。

该选项设计是用各种伺服工具设计而成的，但也可以用于任何类型的附加轴。

优点示例：

- 一台机器人可操作多件工具。
- 由于多件工具在共享一套驱动测量系统，因此需要减少设备。

其中包括

RobotWare选项“伺服工具更换”将使您能够：

- 在线更换工具
- 最多能在8种不同的伺服工具间切换。

请注意，选项“伺服工具更换”仅提供了软件功能，而不包括工具更换器等硬件。

基本方法

这是“伺服工具更换”的一般用法。[第364页的无返回值工具更换程序](#)更详细地说明了其具体做法。

- 1 停用第一件工具。
- 2 从电缆上断开第一件工具。
- 3 将第二件工具接上电缆。
- 4 激活第二件工具。

10 Tool control options

10.1.2 要求与限制

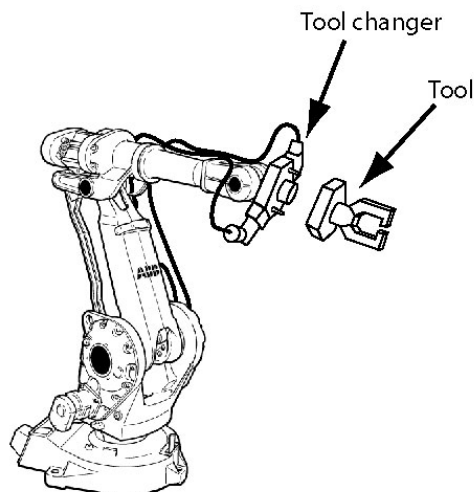
10.1.2 要求与限制

附加轴

您必须拥有选项“附加轴”才能使用“伺服电机控制”。必须按[第360页的要求与限制](#)中的指令来配置“伺服电机控制”使用的所有附加轴。

工具更换器

要想在生产中用某种插件机制来更换工具，就得有一个机械工具更换器接口。



en0300000549

所有电缆均与工具更换器相连。该工具更换器接口包括了信号、电力、空气、水或向 / 由工具输送的任何对象所需的连接件。

最多8种工具

一个机器人控制器上最多同时安装8根附加轴（伺服工具或其它轴）。可让其中的一些（或全部）伺服工具共享同一个工具更换器。

移动已停用的工具

相关控制器会“记住”已停用工具的位置；当重新接入该工具时，系统便会在该位置处激活该工具。

如果在停用期间移动了相关的伺服工具轴，那么该轴在激活后的位置可能有误，而相关控制器却无法检测到这一点。

如果始终未移动该轴，或移动范围不到0.5电机转数，那么激活后的位置就没有问题。



提示

如果您拥有“点焊伺服”（Spot Servo），那么您就能使用工具更换校准。
在激活一件工具后，请调用指令STCalib来校准该工具，以此调节停用期间因工具移动而产生的任何位置错误。

激活错误工具

一项重点就是激活一个未连接的机械单元。

下一页继续

激活错误的机械单元可能会造成意外的运动或错误。如果在未连接任何工具情况下激活了一件工具，那么也会发生相同的错误。



提示

用户可配置连接中继器，从而使系统只能激活已连接的机械单元。具体请参见[第363页的连接中继器](#)。

10 Tool control options

10.1.3 配置

10.1.3 配置

配置概述

选项“伺服工具更换”可为同一附加轴配置多件工具。
每件焊枪工具各安装有一套参数。

如何配置每件工具

像仅有一件工具那样来配置每件工具。具体的做法请参见[第362页的配置](#)。

必须将类型机械单元中参数断开连接时停用PTC监控设置成“是”（Yes）。

必须将类型测量通道中参数断连停用设置成“是”（Yes）。

可将多件工具在类型关节中参数逻辑轴设置成同一数值。由于系统绝不会同时使用这些工具，因此这些工具可以使用相同的逻辑轴。

必须在类型“机械单元”下为具体的伺服焊枪设置参数

`allow_activation_from_any_motion_task`。制造该伺服焊枪时会创建`servo gun .cfg`文件。

对各个参数的详细描述请参见[第362页的配置](#)。

10.1.4 连接中继器

概述

为了确保不会激活已断开的机械单元，用户可以使用一个连接中继器。除非设置了一个指定的数字信号，否则连接中继器就会阻止机械单元被激活。

某些工具更换器支持那些规定当前所接焊枪的I / O信号。连接中继器之后便会使用来自此类工具更换器的一个数字输入信号。

如果工具更换器不支持I / O信号，那么可用RAPID指令创建一种类似行为。每当连接相关工具时，就用指令SetDO将一个数字输出信号设置成1；而当断开该工具时，就将该信号设置成0。

系统参数

此处简述了用于配置一个连接中继器的每个参数。更多信息请参见[第363页的连接中继器](#)中的各个参数。

必须为主题运动下的类型机械单元设置下列参数：

参数	描述
使用连接中继器	待使用的中继器的名称。 对应着类型中继器下参数名称中指定的名称。

必须为主题运动下的类型中继器设置下列参数：

参数	描述
名称	该中继器的名称。 由类型机械单元中参数使用连接中继器使用。
输入信号	所用数字信号的名称，用于指明是否宜允许激活相关的机械单元。

连接中继器配置示例

此例展示了如何为两件焊枪工具配置连接中继器。只有当信号di1为1时才能激活焊枪1，只有当di2为1时才能激活焊枪2。

如果相关工具更换器只会在连接焊枪1时才将di1设置成1，在连接焊枪2时才将di2设置成1，那么就不存在激活错误焊枪的风险。

在类型机械单元中为焊枪1和焊枪2设置下列参数值：

名称	使用连接中继器
焊枪1	gun1_relay
焊枪2	gun2_relay

在类型中继器中为焊枪1和焊枪2设置下列参数值：

名称	输入信号
gun1_relay	di1
gun2_relay	di2

10 Tool control options

10.1.5 无返回值工具更换程序

10.1.5 无返回值工具更换程序

如何更换工具

此处描述了如何将焊枪1更换为焊枪2。

步骤	操作
1	用指令 <code>DeactUnit gun1</code> ;停用焊枪1
2	断开焊枪1与工具更换器之间的连接。
3	将焊枪2与工具更换器连接起来。
4	用指令 <code>ActUnit gun2</code> ;激活焊枪2
5	可选，但建议选用： 用指令 <code>STCalib gun1 \ToolChg</code> ;校准焊枪2 请注意，需要有选项“伺服工具控制”或“点焊伺服”才能进行这种校准。

10.1.6 被禁止激活的点动伺服工具

概述

同一时间只能从工具更换器使用的若干伺服工具中激活一件，而其它工具则被设置成“禁止激活”。这是为了确保用户在按正确的配置来点动当前连接的伺服工具。

禁止激活时该如何行事

当您需要点动一件伺服工具、但因激活被禁用而无法激活相关单元时，则请遵循以下步骤。

步骤	操作
1.	确保在相关工具更换器上安装了正确的伺服工具。若安装了错误的工具，则请参见 第364页的无返回值工具更换程序 。
2.	如果未激活任何工具，那么就打开RAPID执行过程，然后激活正确的工具。
3.	如果在相关工具更换器上安装了正确的伺服工具，那么就停用错误的工具，然后在RAPID执行过程中激活正确的工具。

10 Tool control options

10.2.1 概述

10.2 Tool Control [1180-1]

10.2.1 概述

目的

*Tool Control*可用于控制一个伺服工具，例如在点焊应用中。*Tool Control*可以按特定的板厚度和力度关闭工具，并在此期间保持力度，直到工具被要求再次打开。

其中包括

*Tool Control*可以让您使用：

- 打开、关闭和校准伺服工具的RAPID指令
- 微调系统参数值的RAPID指令
- 检查伺服工具状态的RAPID指令
- 用于配置伺服工具的系统参数

基本方法

这是使用*Tool Control*的常规方法。

- 1 配置和校准伺服工具。
- 2 执行力度校准。
- 3 创建一则RAPID程序。

操作前提

一件伺服工具就是一根附加轴。使用伺服工具的机器人系统上必须有选项*Additional Axes*。应用手册 - *Additional axes and stand alone controller*中规定了驱动模块和测量板等所需硬件。

10.2.2 伺服工具的移动

伺服工具的关闭和打开

伺服工具可以按照预先确定的厚度和工具力度关闭。当工具到达编程设定的接触位置时，移动停止，并立即从位置控制模式切换为力度控制模式。在力度控制模式下，将被施加电机转矩以实现所需的工具力度。

力度会保持恒定，直到收到打开命令。工具打开将会将工具力度减少到零，并将工具臂移动回关闭前的位置。

同步移动与异步移动

一件伺服工具通常会与相关机器人同步移动，使两者能精确地在同一时间完成移动。不过用户也能以异步方式关闭伺服工具（即与当前的机器人移动无关）。这种关闭会使该工具的臂立即开始向预计接触位置（厚度）移动，而这种关闭移动则会中断该工具的臂正在进行的同步移动。

系统也可能在机器人移动时打开相关工具，但如果该机器人的移动中包括了相关伺服工具轴的同步移动，那么就无法做到这一点。此时将发生一次运动错误“无法在与机器人同步移动时打开工具”。

10.2.3 焊枪头管理

关于焊枪头管理

该焊枪头管理功能将自动找出并校准相关工具焊枪头的接触位置，并更新和监测该工具焊枪头的焊枪头总磨损。

焊嘴可以使用RAPID指令STCalib来校准（请参阅第370页的指令：）。通常，在校准期间将执行两次工具关闭。

支持三种不同类型的校准：焊枪头磨损、焊枪头更换和工具更换。所有三种方法都会校准相关焊枪头的接触位置，不过这些方法会以不同方式来更新焊枪头总磨损。

焊枪头磨损校准

当焊嘴磨损后，例如点焊时，则需要处理。焊嘴处理后，需要进行焊嘴磨损校准。工具接触位置会被校准，工具的总体焊嘴磨损将被更新。校准移动很快，并会在零位切换到力度控制模式。

这种方法只能用于焊枪头磨损和 / 或焊枪头打磨后所需的小幅位置调节（<3毫米）。



提示

您RAPID程序中的一个变量可支持跟踪相关焊枪头的磨损，并在需要更换焊枪头时通知您。

焊枪头更换校准

在更换了一对新焊嘴后需要进行焊嘴更换校准，例如点焊时。工具接触位置会被校准，总体焊嘴磨损情况被重置。第一次校准移动较慢，以便发现位置接触位置并切换到力度控制。第二次校准移动较快。这种校准方法将可以处理伺服工具的大位置调整。

为了将相关焊枪头挤压到位，用户可在这种校准后关闭一次工具，然后重新进行一次焊枪头更换校准，以此更新焊枪头挤压后可能出现的位置差异。

工具更换校准

重新连接并激活一件伺服工具后便要进行工具更换校准。系统会校准相关工具的接触位置，而该工具的焊枪头总磨损则保持不变。为了找出未知的焊枪头碰撞位置并切换为力度控制，第一次校准移动的速度较慢，而第二次校准移动的速度则较快。这种校准方法可处理相关工具的大幅位置调节。

由于激活操作会恢复相关工具最后的已知位置，而该位置又可能并非实际的工具位置（该工具的臂在断开连接时可能已移动过），因此每次重新连接一件工具后，都最好采用一下此方法。这种校准方法可处理相关工具的大幅位置调节。



提示

最常见的用法是搭配RobotWare选项“伺服工具更换”来使用工具更换校准。

10.2.4 监控

最大冲程与最小冲程

如果相关工具即将达到最大冲程，或其几乎要接触到焊枪头（即达到最小冲程），那么某种超范围监控就会停止相关运动。具体请参见[第372页的Arm](#)中的*Upper Joint Bound*和*Lower Joint Bound*。

动作监控

在关闭 / 打开的位置控制阶段，系统会激活针对相关伺服工具的运动监控，以检测其臂是否发生碰撞或被卡住。碰撞会带来一次运动错误，使得系统停止相关运动。

在力度控制阶段，运动监控功能会监管相关工具的臂位置，使其不会超出以预计接触位置为起点的特定距离。参见[第373页的Supervision Type](#)中的参数*Max Force Control Position Error*。

最大扭矩

为了避免相关工具受损，相关伺服工具具有一个绝不会超过的最大电机扭矩。如果编写的力度超出了相关工具力度 - 扭矩表中的范围，那么输出力度将被限制在这一最大允许电机扭矩上，同时记录下一条运动警告。具体请参见[第371页的SG Process](#)中的参数*Max Force Control Motor Torque*。

速度限值

力度控制阶段存在着速度限制。即使在相关工具完全关闭前就启动了力度控制，这种速度限制也会使相关工具做出某种受控行为。具体请参见[第372页的Force Master Control](#)中的*Speed limit 1- 6*。

10 Tool control options

10.2.5 RAPID组件

10.2.5 RAPID组件

关于RAPID组件

这是 *Tool Control* 中所有指令、函数和数据类型。

有关更多信息，请参阅 技术参考手册 - *RAPID* 指令、函数和数据类型。

指令：

指令	描述
STClose	按预定义的力度和厚度来关闭相关的伺服工具。
STOpen	打开伺服工具。
STCalib	校准伺服工具。 某个调用函数将决定执行哪一类校准： <ul style="list-style-type: none">• 用于工具更换校准的 \ToolChg• 用于焊枪头更换校准的 \TipChg• 用于焊枪头磨损校准的 \TipWear
STTune	微调相关伺服工具的各个运动参数。可为相关指令所指定的参数设置一个临时值。
STTuneReset	为相关伺服工具重置已微调的各个运动参数。撤销所有 STTune 指令的作用。

函数

功能	描述
STIsClosed	测试是否关闭了相关的伺服工具。
STIsOpen	测试是否打开了相关的伺服工具。
STIsCalib	测试是否校准了相关的伺服工具。
STCalcTorque	计算一件伺服工具的电机扭矩。
STCalcForce	计算一件伺服工具的力度。
STIsServoTool	测试某个机械单元是否为一件伺服工具。
STIsIndGun	测试伺服工具是否处于独立模式。

数据类型

Tool Control 不包含 RAPID 数据类型。

10.2.6 系统参数

关于系统参数

在使用伺服工具时，相关控制器上通常会装有关于该工具的运动参数文件。一件伺服工具就是一根附加轴的专用改型，而应用手册 - *Additional axes and stand alone controller* 则描述了如何配置这种伺服工具。

在本节简要介绍了与 *Tool Control* 搭配使用的参数。如需了解更多信息，请参阅技术参考手册 - 系统参数中的相应参数。

SG Process

这些参数属于主题 *Motion* 下的类型 *SG Process*。

SG Process 的作用是配置一支伺服焊枪（或其它伺服工具）的行为。

参数	描述
<i>Close Time Adjust</i>	调节相关焊枪的预定最短关闭时间。
<i>Close Position Adjust</i>	关闭相关焊枪时对启动力度控制的预定位置（板厚）的调节。
<i>Force Ready Delay</i>	在达到预定力度后延迟相关的关闭就绪事件。
<i>Max Force Control Motor Torque</i>	用于力度控制的最大允许电机扭矩。如果所需的电机扭矩高于该数值，那么系统就会减少所命令的力度。
<i>Post-synchronization Time</i>	对打开就绪事件的预期。可由此让“打开焊枪”与“下一次机器人移动”同步。
<i>Calibration Mode</i>	定义在焊枪头磨损校准期间关闭相关伺服焊枪的次数。
<i>Calibration Force Low</i>	焊枪头磨损校准期间采用的最小焊枪头力度。
<i>Calibration Force High</i>	焊枪头磨损校准期间采用的最大焊枪头力度。
<i>Calibration Time</i>	校准时相关伺服焊枪在已关闭位置等候的时间。
<i>Number of Stored Forces</i>	在 <i>Tip Force 1 - 10</i> 和 <i>Motor Torque 1 - 10</i> 规定的力度 - 扭矩关系中定义相应的点数。
<i>Tip Force 1 - 10</i>	<i>Tip Force 1</i> 定义了 <i>Motor Torque 1</i> 中电机扭矩所对应的焊枪头力度。 <i>Tip Force 2</i> 对应着 <i>Motor Torque 2</i> 等。
<i>Motor Torque 1 - 10</i>	<i>Motor Torque 1</i> 定义了 <i>Tip Force 1</i> 中焊枪头力度所对应的电机扭矩。 <i>Motor Torque 2</i> 对应着 <i>Tip Force 2</i> 等。
<i>Squeeze Position 1 - 10</i>	定义了相关力度校准表中每个力度等级的关节位置。
<i>Soft Stop Timeout</i>	定义了当恒力期间发生软停止时，将保持多久的相关力度。

Force Master

这些参数属于主题 *Motion* 下的类型 *Force Master*。

Force Master 用于定义伺服工具（通常是焊枪）在力度控制期间的行为方式。此参数仅影响力度控制模式下的伺服工具。

参数	描述
<i>References Bandwidth</i>	针对各引用值的低通滤波器的频率限值。
<i>Use ramp time</i>	确定增减相关焊枪头力度时是否宜使用某种恒定时间或恒定梯度。
<i>Ramp when Increase Force</i>	决定当 <i>Use ramp time</i> 被设置成“否”（No）时，在关闭相关工具的同时要以多快的速度加大力度。

下一页继续

10 Tool control options

10.2.6 系统参数

续前页

参数	描述
<i>Ramp time</i>	决定当 <i>Use ramp time</i> 被设置成“是” (Yes) 时，在关闭相关工具的同时要以多快的速度加大力度。
<i>Collision LP Bandwidth</i>	针对焊枪头磨损校准的低通滤波器的频率限值。
<i>Collision Alarm Torque</i>	决定了当焊枪首次结束新焊枪头校准和工具改动校准时，相应的各工具焊枪头要共同施加多大的压力。
<i>Collision Speed</i>	决定了焊枪首次结束新焊枪头校准和工具改动校准时的伺服焊枪速度。
<i>Collision Delta Position</i>	定义了当相关电机扭矩达到 <i>Collision Alarm Torque</i> 中指定的数值时，相关伺服工具超过接触位置后继续移动的距离。
<i>Max pos err. closing</i>	决定了在启动力度控制前，相关工具的焊枪头必须距离预定板厚多远。
<i>Delay ramp</i>	在启动力度控制时延后启动扭矩增减。
<i>Ramp to real contact</i>	确定当决定相关的接触位置时，是否宜用反馈位置来取代参考位置。

Force Master Control

这些参数属于主题*Motion*下的类型*Force Master Control*。

*Force Master Control*的作用是将速度限值和速度环路增益设置成相关扭矩的函数。

参数	描述
<i>No. of speed limits</i>	用于将速度限值和速度环路增益定义为相关扭矩之函数的点数。最多可定义6个点。
<i>torque 1 - torque 6</i>	扭矩等级，对应着定义了相关速度限值和速度环路增益值的预定焊枪头力度。
<i>Speed Limit 1 - 6</i>	使用 <i>Speed Limit 1</i> 到 <i>Speed Limit 6</i> 来定义取决于预定焊枪头力度的最大速度。
<i>Kv 1 - 6</i>	使用 <i>Kv 1</i> 到 <i>Kv 6</i> 来定义速度环路增益，以供在超出相关速度限值时降低速度。

Arm

这些参数属于主题*Motion*下的类型*Arm*。

类型*Arm*定义了一支臂的各种特征。

参数	描述
<i>Upper Joint Bound</i>	定义了相关关节工作区域的上限。
<i>Lower Joint Bound</i>	定义了相关关节工作区域的下限。

Acceleration Data

这些参数属于主题*Motion*下的类型*Acceleration Data*。

*Acceleration Data*的作用是为无任何动态模型的轴指定某些加速度特征。

参数	描述
<i>Nominal Acceleration</i>	最坏情况下的电机加速度。
<i>Nominal Deceleration</i>	最坏情况下的电机减速度。
<i>Acceleration Derivate Ratio</i>	指明加速度的增大速度。
<i>Deceleration Derivate Ratio</i>	指明减速度的增大速度。

下一页继续

Motor Type

这些参数属于主题*Motion*下的类型*Motor Type*。

Motor Type 的作用是描述一部电机的特征。

参数	描述
<i>Pole Pairs</i>	定义相关电机的双极对数目。
<i>Inertia</i>	相关电机（包括旋转变压器，但不包括制动器）的惯量。
<i>Stall Torque</i>	连续失速扭矩，即相关电机在永远无速度时产生的扭矩。
<i>ke Phase to Phase</i>	标称电压常数。对应着速度1弧度 / 秒的感应电压（相间电压）。
<i>Max Current</i>	没有不可逆磁化时的最大电流。
<i>Phase Resistance</i>	每一相在20摄氏度下的标称绕组电阻。
<i>Phase Inductance</i>	每一相在零电流下的标称绕组电阻。

Motor Calibration

这些参数属于主题*Motion*下的类型*Motor Calibration*。

*Motor Calibration*的作用是校准一部电机。

参数	描述
<i>Commutator Offset</i>	定义了当转子位于相对于定子的电气零位时，相关电机（旋转变压器）所处的位置。
<i>Calibration Offset</i>	定义了臂处于校准位置时，相关电机（旋转变压器）所处的位置。

Stress Duty Cycle

这些参数属于主题*Motion*下的类型*Stress Duty Cycle*。

*Stress Duty Cycle*的作用是保护轴和齿轮箱等。

参数	描述
<i>Speed Absolute Max</i>	待使用的绝对最高电机速度。
<i>Torque Absolute Max</i>	待使用的绝对最高电机扭矩。

Supervision Type

这些参数属于主题*Motion*下的类型*Supervision Type*。

*Supervision Type*的作用是持续监控位置、速度和扭矩。

参数	描述
<i>Max Force Control Position Error</i>	当一支伺服焊枪处于力度控制模式时，其移动距离不得超过 <i>Max Force Control Position Error</i> 中指定的距离。如果发生了丢失某一焊枪头之类的情况，此监控功能便会保护相关工具。
<i>Max Force Control Speed Limit</i>	力度控制期间的速度误差系数。 如果超出了类型 <i>Force Master Control</i> 中定义的速度限值（与 <i>Max Force Control Speed Limit</i> 叠加后），那么系统就会停止所有移动。

下一页继续

10 Tool control options

10.2.6 系统参数

续前页

Transmission

这些参数属于主题*Motion*下的类型*Transmission*。

*Transmission*的作用是定义一部电机与其轴之间的传动齿轮比。

参数	描述
<i>Rotating Move</i>	定义相关的轴是旋转轴还是线性轴。
<i>Transmission Gear Ratio</i>	定义电机与关节之间的传动齿轮比。

Lag Control Master 0

这些参数属于主题*Motion*下的类型*Lag Control Master 0*。

*Lag Control Master 0*的作用是调整无任何动态模型的轴。

参数	描述
<i>FFW Mode</i>	定义相关的位置调整中是否宜使用速度值和扭矩值前馈。
<i>Kp, Gain Position Loop</i>	位置调整环路中的比例增益。
<i>Kv, Gain Speed Loop</i>	速度调整环路中的比例增益。
<i>Ti Integration Time Speed Loop</i>	速度调整环路中的积分时间。

Uncalibrated Control Master 0

这些参数属于主题*Motion*下的类型*Uncalibrated Control Master 0*。

*Uncalibrated Control Master 0*的作用是调整未经校准的轴。

参数	描述
<i>Kp, Gain Position Loop</i>	位置调整环路中的比例增益。
<i>Kv, Gain Speed Loop</i>	速度调整环路中的比例增益。
<i>Ti Integration Time Speed Loop</i>	速度调整环路中的积分时间。
<i>Speed Max Uncalibrated</i>	一根未校准的轴的最大允许速度。
<i>Acceleration Max Uncalibrated</i>	一根未校准的轴的最大允许加速度。
<i>Deceleration Max Uncalibrated</i>	一根未校准的轴的最大允许减速度。

10.2.7 调试与服务

调试伺服工具

按以下步骤来安装和调试新的伺服工具：

步骤	操作
1	按应用手册 - <i>Additional axes and stand alone controller</i> 中的说明来安装相关的伺服工具。
2	载入带有伺服工具配置的一份.cfg文件。操作员手册 - <i>RobotStudio</i> 更详细地说明了其具体做法。 如果您没有任何针对相关伺服工具的.cfg文件，那么可以载入一份模板文件，然后按您伺服工具的数值来配置相应的系统参数。用户可在RobotWare发行版中找到各种模板文件。具体请参见 第375页的模板文件的位置 。
3	使用RAPID指令STTune，并迭代找出最佳参数值，然后宜在相关系统参数中永久写入这些最佳值。
4	精细校准伺服工具。具体请参见 第376页的精细校准 。
5	除非在加载的.cfg文件中包含力度校准，否则都需要进行力度校准。

模板文件的位置

可从相应的PC或IRC5控制器处获得相应的模板文件。

- 在RobotStudio的RobotWare安装文件夹中：...\\RobotPackages\\RobotWare_RPK_<version>\\utility\\AdditionalAxis\\
- 在相应的IRC5控制器上：
<SystemName>\\PRODUCTS\\<RobotWare_xx.xx.xxxx>\\utility\\AdditionalAxis\\



注意

右键单击插件浏览器中的已安装RobotWare版本，然后选择打开数据包文件夹，系统便会从RobotStudio插件标签处将您引导至RobotWare安装文件夹。

断开 / 重重新连接一件伺服工具

如果停用了相关的伺服工具，那么或可用DeactUnit指令来断开并移除该工具。当连接并重新激活该工具时，系统又会恢复该工具在停用时的位置。做一次工具更换校准，以此来确保焊枪头位置正常。

如果您使用了RobotWare的选项Servo Tool Change和指令STCalib，那么便可用RAPID程序来执行整个工具更换过程。

意外断开连接后的恢复

在激活伺服工具的情况下，如果意外断开了相关电机的电缆，那么系统就会陷入系统错误状态。在重启系统后，用户必须停用相应的伺服工具，以便将相关机器人点动到某个服务位置处。

用户可在点动窗口执行停用操作。轻击激活...，再选择相关的伺服工具，然后轻击停用。

经过保养 / 修理后，由于已经丢失了相关位置，所以必须更新相应的转数计数器。具体请参见[第376页的更新转数计数器](#)。

10.2.8 机械单元的校准

精细校准

在安装新的伺服工具或伺服工具轴处于“未校准”状态时，必须进行微校。

对此，建议使用下列指令创建一个服务例行程序：

```
STCalib "ToolName" \TipChg;  
STCalib "ToolName" \TipWear;
```

更新转数计数器

如果轴位置丢失，则必须执行转数计数器更新。如果发生此情况，则会有校准状态“转数计数器未更新”。

对此，建议使用与微校相同的服务例行程序。

10.2.9 RAPID代码示例

如何使用代码包

通常的*Tool Control*编程技巧是根据下面的示例代码自定义外壳例行程序。然后从您的程序调用这些外壳例行程序。

使用壳例程

此例展示了与使用标准伺服工具指令的某定制例程（`rMoveSpot`）相结合的一则主例程。其中使用了例程`rWeld`来表示外部进程（比如焊接计时器）。

```
PROC main()
  MoveJ p1, v500, z50, weldtool;
  MoveL p2, v1000, z50, weldtool;
  ! Perform weld process
  rMoveSpot weldpos1, v2000, curr_gun_name, 1000, 2, 1,
    weldtool\WObj:=weldwobj;
  rMoveSpot weldpos2, v2000, curr_gun_name, 1000, 2, 1,
    weldtool\WObj:=weldwobj;
  rMoveSpot weldpos3, v2000, curr_gun_name, 1500, 3, 1,
    weldtool\WObj:=weldwobj;
  MoveL p3, v1000, z50, weldtool;
ENDPROC

PROC rMoveSpot (robtarg ToPoint,
  speeddata Speed,
  gunname Gun,
  num Force,
  num Thickness,
  PERS tooldata Tool
  \PERS wobjdata WObj)
  ! Move the gun to weld position.
  ! Always use FINE point to prevent too early closing.
  MoveL ToPoint, Speed, FINE, weldtool \WOIbj=WObj;
  STCloseGun Gun, Thickness;
  rWeld;
  STOpenGun Gun;
ENDPROC

PROC rWeld()
  ! Request weld start from weld timer
  SetDO doWeldstart,1;
  ! Wait until weld is performed
  WaitDI diWeldready,1;
  SetDO doWeldstart,0;
ENDPROC
```

10.3 I/O Controlled Axes [包含在1180-1中]

10.3.1 概述

目的

*I/O Controlled Axes*旨在使用I/O接口而非将轴集成到IRC5驱动系统从机器人控制器控制轴。

对于操作和编程，I/O控制的轴与集成加工轴表现相同。区别在于I/O控制的轴不直接连接到机器人控制器的驱动系统。动作配置提供了一个I/O接口，该接口将机器人控制器连接到外部伺服调节器。

机器人控制器可以在程序执行期间获取和释放附加轴的控制。附加轴可以与机器人（由机器人控制器控制时）同步移动，或独立于机器人（由外部PLC控制时）移动。

一些应用示例：

- 伺服焊枪
- 夹具

其中包括

RobotWare选项*I/O Controlled Axes*让您可以使用系统参数来配置I/O控制的轴。

基本方法

这是设置*I/O Controlled Axes*的常规方法。

- 1 配置要通过I/O控制的轴的系统参数。请参阅[第383页的配置](#)。
- 2 按对待其他附加轴一样操作轴（手动操纵、程序操纵等）。请参阅[第387页的RAPID编程](#)。

对于附加轴的一般情况，也请参阅操作员手册 - 带 *FlexPendant* 的 *IRC5* 和应用手册 - *Additional axes and stand alone controller*。

10.3.2 轮廓错误

什么是轮廓错误

如果`robtarget`编程设定的机器人路径上的I/O控制轴未按照总线延迟和加速到达，则会生成轮廓错误。如果发生此事件，机器人的移动会在路径上停止。在错误日志中会产生一条错误记录。

轮廓错误发生的可能原因：

- 机器人碰撞
- 难以移动或故障的外部轴
- 系统参数*Bus delay time in ms*值不正确

错误处理

1 错误 – 外部处理部件的确认。

对此，每个应用程序都需要提供“重置”按钮。处理部件需要在程序可以启动前就绪。

2 电机上电/程序启动

如果允许自动移动回路，机器人将会在程序继续前自动返回到指令被取消时路径的位置上。如果不允许自动移动，则会产生一个错误消息。会有选择菜单供选择接受移动或取消启动事件。

启动事件取消时，操作者需要将操作模式修改为手动。

现在操作者可以在机器人程序可以重启前指定后续程序。例如：

- 手动将机器人移出碰撞区域
- 按此前的移动指令移动

如需了解更多信息，请参阅技术参考手册 - 系统参数中的主题*Controller*，类型*Path Return Region*。

10.3.3 纠正位置

纠正位置

纠正（教导）机器人位置(`robtarg`)是使用程序编辑器中的修改位置按钮（对于机器人轴）来进行的。

对于下列状态，I/O控制轴的修改后位置将不会是正确的位置，而是最后的有效反馈位置：

- 轴未被引用
- 伺服调节器不工作
- I/O接口的实际位置无效
- 位置在操作范围之外

仅激活的轴采用位置纠正。如果可用轴未激活，则忽略该轴。这意味着`robtarg`替代该轴的符号保持不变。这种状态不会导致错误。

10.3.4 工具变换

工具变换

如果使用指令`DeactUnit`停用工具，则有必要将信号设置为禁用。当工具禁用（可以通过信号`unit_disabled`验证），则有可能断开工具的供电，例如卸下一个点焊枪。不同的工具可以配置相同的逻辑轴编号，但是这要求RobotWare选项*Servo Tool Change*。

10.3.5 安装

安装

安装了机器人系统后，I/O控制的轴需要在系统参数中加载。

每个要求的轴都需要分别加载。具体的动作文件包括默认的动作参数。加载轴的参数化与调节在配置中详细介绍。

10.3.6 配置

默认设定

带有选项 *I/O Controlled Axes* 的机器人系统将会默认有一个名为 EXTCTL1 的机械单元。这将成为 7 号逻辑轴。

对于 EXTCTL1，将在类型 *External Control Process Data* 和主题 *Motion* 中定义默认信号名称。

对 I/O 控制轴的强制设置

对于应该用作 I/O 控制轴的机械单元，必须用数据完成下列配置。

- 1 在类型 *Transmission*，设置 *Transmission Gear Ratio*。请参阅第 386 页的类型 *Transmission*。
- 2 在类型 *Acceleration Data*，设置 *Nominal Acceleration*、*Nominal Deceleration*、*Acceleration Derivate Ratio* 和 *Deceleration Derivate Ratio*。请参阅第 385 页的类型 *Acceleration Data*。
- 3 在类型 *Arm*，设置 *Upper Joint Bound* 和 *Lower Joint Bound*。请参阅第 385 页的类型 *Arm*。
- 4 在类型 *Stress Duty Cycle*，设置 *Speed Absolute Max*。请参阅第 386 页的类型 *Stress Duty Cycle*。
- 5 在类型 *Supervision Type*，设置 *static_position_limit* 和 *dynamic_position_limit*。请参阅第 386 页的类型 *Supervision Type*。
- 6 在类型 *External Control Process Data*，设置 *Bus delay time in ms*。请参阅第 385 页的类型 *External Control Process Data*。

可选自定义设置

如果默认值之外的其他值更优，下列任意设置均可以修改。

- 要使用 7 号之外的逻辑轴，修改 *Logical Axis* 的值。请参阅第 386 页的类型 *Joint*。
- 要修改用于与 I/O 控制轴通信的信号名称，请修改类型 *External Control Process Data* 中的设置。请参阅第 385 页的类型 *External Control Process Data*。
- 要使用启动继电器，请设置参数 *Use Activation Relay*。请参阅第 386 页的类型 *Mechanical Unit*。

添加其他轴

对于第二个 I/O 控制轴，必须加载一个配置文件。

- 1 从 IRC5 控制器 <system name> \PRODUCTS\RobotWare_6.XX.XXXX\options\ioctrlaxis\ 在 RobotStudio 或 FlexPendant 中加载一个 .cfg 文件。
- 2 进行与第一个 I/O 控制轴相同的配置。



注意

多个机械单元可以使用同一个逻辑轴编号，但是这要求 RobotWare 选项 *Servo Tool Change*。

下一页继续

10 Tool control options

10.3.6 配置

续前页

PROFINET的设置

如果使用了PROFINET总线，I/O控制部件的参数*Reduction ratio*应设置为4 ms或2。
请参阅应用手册 - *PROFINET Controller/Device*。

10.3.7 系统参数

关于系统参数

这是对I/O Controlled Axes中每个参数的简要说明。如需了解更多信息，请参阅技术参考手册 - 系统参数中的相应参数。

类型External Control Process Data

这些参数属于*External Control Process Data*类，在主题 *Motion*中。

参数	描述
Bus delay time in ms	总线延时参数
Regulator activation signal	I/O控制单元启动输出信号。
Ext Controller output signal	允许对单元进行外部控制的输出信号
Pos_ref output signal	带I/O控制轴定位参考点的输出信号
Pos_ref sign signal	带I/O控制轴定位参考标志（+或-）的输出信号
Pos_ref valid signal	指示定位参考点为有效信号并且轴需要遵循参考信号的输出信号。
Regulator is activated signal	指示I/O控制单元是否启用并且准备就绪的输入信号。
Req pos is out of range input signal	指示必要定位参考点是否超出范围的输入信号。
Pos_fdb input signal	带I/O控制轴位置反馈的输入信号
Pos_fdb sign signal	带I/O控制轴位置反馈标志（+或-）的输入信号。
Pos_fdb_valid signal	指示位置反馈信号有效的输入信号。
Unit_ready input signal	来自I/O控制单元来指示I/O控制单元已就绪的输入信号。
Ext Controller input signal	指示由外部装置控制移动的输入信号。不允许机器人控制柜移动此外部装置。

类型Acceleration Data

这些参数属于*Acceleration Data*类，在主题 *Motion*中。

参数	描述
Nominal Acceleration	最坏情况下的电机加速度。
Nominal Deceleration	最坏情况下的电机减速度。
Acceleration Derivate Ratio	定义加速变化的速度，即加速度的变化率指标。
Deceleration Derivate Ratio	定义减速变化的速度，即减速度的变化率指标。

类型Arm

这些参数属于主题*Motion*下的类型*Arm*。

参数	描述
Upper Joint Bound	定义此关节工作区域的上限。
Lower Joint Bound	定义此关节工作区域的下限。

下一页继续

10 Tool control options

10.3.7 系统参数

续前页

类型Joint

这些参数属于*Joint*类，在主题 *Motion*中。

参数	描述
Logical Axis	定义RAPID程序看到的轴编号。 两个机械单元可以在 <i>Logical Axis</i> 上使用相同的值设置，但不能同时启动。

类型Mechanical Unit

这些参数属于*Mechanical Unit*类，在主题 *Motion*中。

参数	描述
Use Activation Relay	指出当机械单元启动或停止时将启动或停止的继电器。

类型Stress Duty Cycle

这些参数属于*Stress Duty Cycle*类，在主题 *Motion*中。

参数	描述
Speed Absolute Max	待使用的绝对最高电机速度。

类型Supervision Type

这些参数属于*Supervision Type*类，在主题 *Motion*中。

参数	描述
static_position_limit	电机侧弧度内零速度下的位置错误限制。
dynamic_position_limit	电机侧弧度内最大速度下的位置错误限制（最大延迟）。

类型Transmission

这些参数属于*Transmission*类，在主题 *Motion*中。

参数	描述
Transmission Gear Ratio	定义电机与关节之间的传动齿轮比。

10.3.8 RAPID编程

数据类型

这是对于使用I/O Controlled Axes时的RAPID数据类型的特别注意事项的简要说明。
数据类型的一般说明请参阅技术参考手册 - *RAPID*指令、函数和数据类型。

数据类型	描述
robtarget	I/O控制轴的位置设置为robtarget上的一个附加轴。 例如，对于I/O控制周为逻辑轴7且应该移动到位置100： <pre>p1 := [[20,50,-80], [1,0,0,0], [1,1,0,0], [100,9E+09,9E+09,9E+09,9E+09,9E+09]];</pre>

指令：

这是对于使用I/O Controlled Axes时有关RAPID指令的特别注意事项的简要说明。
指令的一般说明请参阅技术参考手册 - *RAPID*指令、函数和数据类型。

指令	描述
MoveL MoveC MoveJ	用于移动I/O控制轴的一般移动指令。I/O控制值的位置值包含在robtarget中，请参阅 第387页的数据类型 。 I/O控制周可以与机器人同步移动。

RAPID示例

```
PROC Sequence123()
...
MoveJ pHome, v1500, fine, tGun1;
ActUnit EXTCTL1;
MoveJ p100, v1000, z10, tGun1 \Wobj:=wobj1;
MoveL p101, v1000, fine, tGun1 \Wobj:=wobj1;
...
! Application-specific commands
...
MoveL p102, v1000, z10, tGun1 \Wobj:=wobj1;
MoveJ p100, v1000, fine, tGun1 \Wobj:=wobj1;
DeactUnit EXTCTL1;
MoveJ pHome, v1500, fine, tGun1;
ENDPROC
```

此页刻意留白

索引

A

Absolute Accuracy, 111
 Absolute Accuracy补偿, 120
 Absolute Accuracy验证, 123
 Acceleration Data, 372, 383, 385
 Acceleration Derivate Ratio, 372, 385
 Acceleration Max Uncalibrated, 374
 Advanced RAPID, 17
 Advanced Shape Tuning, 134
 AliasIO, 25
 Analog Signal Interrupt, 47
 Analog Synchronization, 155
 ArgName, 45
 Arm, 372, 383, 385
 Auto acknowledge input, 13, 50

B

BitAnd, 19
 BitCheck, 19
 BitClear, 19
 BitLSh, 19
 BitNeg, 19
 BitOr, 19
 BitRSh, 19
 BitSet, 19
 BitXOr, 19
 BookErrNo, 41
 bool, 344
 Bus delay time in ms, 385
 byte, 19
 ByteToStr, 19

C

C# API, 323
 Calibration Force High, 371
 Calibration Force Low, 371
 Calibration Mode, 371
 Calibration Offset, 373
 Calibration Pendulum, 113
 Calibration Time, 371
 CalibWare, 113
 Check unresolved references, Task type, 283
 CirPathMode, 150
 ClearIOBuff, 79
 ClearRawBytes, 83
 Close, 79
 CloseDir, 87
 Close position adjust, 371
 Close time adjust, 371
 Collision Alarm Torque, 372
 Collision Delta Position, 372
 Collision Detection Memory, 244
 Collision Error Handler, 244
 Collision LP Bandwidth, 372
 Collision Speed, 372
 Commutator Offset, 373
 configuration.xml, 349
 CopyFile, 87
 CopyRawBytes, 83
 CorrClear, 236
 CorrCon, 236
 corrdescr, 236
 CorrDiscon, 236
 CorrRead, 236

CorrWrite, 236
 CPU_load_equalization, 204
 Cyclic bool, 51

D

datapos, 22
 Deceleration Derivate Ratio, 372, 385
 Deceleration Max Uncalibrated, 374
 Delay ramp, 372
 description.xml, 347
 dir, 87
 displacement, 68
 dynamic_position_limit, 386

E

EGM, 310
 EGM.proto文件, 339
 EGM Path Correction, 310
 EGM Position Guidance, 310
 EGMRAPID部件, 328
 EGM传感器协议, 323
 EGM执行状态, 315
 EGM系统参数, 327
 Electronically Linked Motors, 57
 errdomain, 38
 ErrRaise, 38
 errtype, 38
 Event Preset Time, 74
 Ext Controller input signal, 385
 Ext Controller output signal, 385
 External Control Process Data, 383, 385
 Externally Guided Motion, 310
 External Motion Interface Data, 327

F

FeedbackJoints, 343
 FeedbackPose, 343
 FeedbackTime, 343
 FFW Mode, 374
 Fieldbus Command Interface, 90
 FIFO, 272
 FileSize, 87
 FlexPendant示教器, 298
 Follower to Joint, 58
 Force Master, 371
 Force Master Control, 372
 Force Ready Delay, 371
 frame, 344
 FricIdEvaluate, 140
 FricIdInit, 140
 FricIdSetFricLevels, 140
 Friction FFW Level, 138
 Friction FFW On, 138
 Friction FFW Ramp, 138
 FSSize, 87

G

General RAPID, 244
 GetDataVal, 22
 GetMaxNumberOfCyclicBool, 53
 GetNextCyclicBool, 53
 GetNextSym, 22
 GetNumberOfCyclicBool, 53
 GetTrapData, 38
 Google C++, 323
 Google C++ API, 323
 Google overview, 323

Google Protocol Buffers, 323

I

I/O Controlled Axes, 378

IError, 38

IIRFFP, 203

IndAMove, 217

IndCMove, 217

IndDMove, 217

Independent Joint, 216

Independent Lower Joint Bound, 216

Independent Upper Joint Bound, 216

IndInpos, 217

IndReset, 217

IndRMove, 217

IndSpeed, 217

Inertia, 373

iodev, 79

IPers, 38

IP协议, 342

IRMQMessage, 275

IsFile, 87

ISignalAI, 48

ISignalAO, 48

IsStopStateEvent, 45

IVarValue, 306

J

Jog Collision Detection, 244

Jog Collision Detection Level, 244

joint, 344

Joint, 58, 383, 386

K

ke Phase to Phase, 373

Kp, Gain Position Loop, 374

Kv 1 - 6, 372

Kv, Gain Speed Loop, 374

Kv, Gain Speed Loop, 374

L

l_f_axis_name, 68

l_f_axis_no, 68

l_f_mecunt_n, 68

l_m_axis_no, 68

l_m_mecunt_n, 68

Lag Control Master 0, 374

Linked M Process, 58

Local path, 253

Lock Joint in Ipol, 58

Logical Axis, 386

Logical Cross Connections, 94

Lower Joint Bound, 372, 385

LTAPP, 305

M

Main entry, Task type, 283

MakeDir, 87

Manipulator Supervision, 244

Manipulator Supervision Level, 244

Master Follower kp, 58

Max Current, 373

Max Follower Offset, 58

Max Force Control Motor Torque, 371

Max Force Control Position Error, 373

Max Force Control Speed Limit, 373

Max Offset Speed, 58

Max pos err. closing, 372

Mechanical Unit, 383, 386

Motion Planner, 244

Motion Process Mode, 141

MotionSup, 245, 249

Motion Supervision, 244

Motion Supervision Max Level, 244

MotionTask, Task type, 283

Motor Calibration, 373

Motor Torque 1- 10, 371

Motor Type, 373

MotSupOn, 246

MotSupTrigg, 246

MoveC, 387

MoveCSync, 74

MoveJ, 387

MoveJSync, 74

MoveL, 387

MoveLSync, 74

Multitasking, 281

N

Name, 253

Name, Transmission Protocol type, 304–305

Nanopb, 323

NFS Client, 255

No. of speed limits, 372

Nominal Acceleration, 372, 385

Nominal Deceleration, 372, 385

NORMAL, 283

NoSafety, 283

NOT, 96

num, 344

Number of Stored Forces, 371

O

offset_ratio, 68

Offset Adjust Delay Time, 58

Offset Speed Ratio, 58

Open, 79

OpenDir, 87

OperationMode, 343

P

PackDNHeader, 91

PackRawBytes, 83

Password, 253

Path Collision Detection, 244

Path Collision Detection Level, 244

pathrecid, 221

PathRecMoveBwd, 221

PathRecMoveFwd, 221

PathRecStart, 221

PathRecStop, 221

PathRecValidBwd, 221

PathRecValidFwd, 221

PC Interface, 259

PC SDK客户端, 271

PFRestart, 31

Phase Inductance, 373

Phase Resistance, 373

PlannedJoints, 343

PlannedPose, 343

Pole Pairs, 373

Pos_fdb_valid signal, 385

Pos_fdb input signal, 385

Pos_fdb sign signal, 385

Pos_ref output signal, 385
 Pos_ref sign signal, 385
 Pos_ref valid signal, 385
 pose, 344
 Post-synchronization Time, 371
 PredictedTime, 343
 Process, 58
 Protobuf, 323
 Protobuf-csharp, 323
 Protobuf-net, 323

R

r1_calib, 114
 Ramp time, 372
 Ramp Time, 58
 Ramp to real contact, 372
 Ramp when Increase Force, 371
 RAPID变量, 340
 RAPID消息队列, 270
 RAPID组件
 传感器接口, 306
 多任务化, 284
 高级RAPID语言, 45
 RAPID编辑器, 262
 RAPID配套功能, 44
 RAPID限制, Machine Synchronization, 181
 rawbytes, 83
 RawBytesLen, 83
 ReadAnyBin, 79
 ReadBin, 79
 ReadBlock, 306
 ReadCfgData, 28
 ReadDir, 87
 ReadErrData, 38
 ReadNum, 79
 ReadRawBytes, 83
 ReadStr, 79
 ReadStrBin, 79
 ReadVar, 306
 real, 344
 References Bandwidth, 371
 Regulator activation signal, 385
 Regulator is activated signal, 385
 Remote Address, 305
 RemoveAllCyclicBool, 53
 RemoveCyclicBool, 53
 RemoveDir, 87
 RemoveFile, 87
 RenameFile, 87
 Req pos is out of range input signal, 385
 restartdata, 34
 RestoPath, 221
 Rewind, 79
 RMQFindSlot, 275
 RMQGetMessage, 275
 RMQGetMsgData, 275
 RMQGetMsgHeader, 275
 RMQGetSlotName, 275
 rmqheader, 275
 rmqmessage, 275
 RMQSendMessage, 275
 RMQSendWait, 275
 rmqslot, 275
 RMQ最大消息大小, 274
 RMQ最大消息数目, 274
 RMQ模式, 274
 RMQ清空队列, 275

RMQ类型, 274
 RMQ读取等候, 275
 RoboCom Light, 305
 RobotStudio, 262
 robtarget, 387
 Rotating Move, 374
 RTP1协议, 304

S

SCWrite, 260
 SEMISTATIC, 283
 SenDevice, 306
 sensor_speed, 180
 Sensor Interface, 302
 Sensor Synchronization, 155
 Serial Port, Transmission Protocol type, 304–305
 Server address, 253
 Server path, 253
 SetAllDataVal, 22
 SetDataSearch, 22
 SetDataVal, 22
 SetSysData, 45
 settings.xml, 346
 SetupCyclicBool, 53
 SG Process, 371
 shapedata, 211
 SiTool, 351
 SiWobj, 351
 Socket Messaging, 263
 Soft Stop Timeout, 371
 speed_ratio, 68
 Speed Absolute Max, 373, 386
 Speed Limit 1 - 6, 372
 Speed Max Uncalibrated, 374
 Squeeze Position 1 -10, 371
 Stall Torque, 373
 STATIC, 283
 static_position_limit, 386
 STCalcForce, 370
 STCalcTorque, 370
 STCalib, 370
 STClose, 370
 StepBwdPath, 34
 STIsCalib, 370
 STIsClosed, 370
 STIsIndGun, 370
 STIsOpen, 370
 STIsServoTool, 370
 STOpen, 370
 StorePath, 221
 Stress Duty Cycle, 373, 383, 386
 string, 344
 StrToByte, 19
 STTune, 370
 STTuneReset, 370
 Supervision Type, 373, 383, 386
 syncident, 294
 syncident, data type, 284
 SyncMoveResume, 221
 SyncMoveSuspend, 221
 SysFail, 283
 SysHalt, 283
 SysStop, 283

T

Task, Task type, 283
 Task, type, 283

taskid, 300
taskid, data type, 284
Task in foreground, 287
Task in foreground, Task type, 283
TaskRunMec, 299
TaskRunMec, function, 284
TaskRunRob, 299
TaskRunRob, function, 284
tasks, 294
 data type, 284
tasks, data type, 284
TestAndSet, 298
TestAndSet, function, 284
TextGet, 41
TextTabFreeToUse, 41
TextTabGet, 41
TextTabInstall, 41
Ti Integration Time Speed Loop, 374
time, 344
Tip Force 1 - 10, 371
torque 1 - torque 6, 372
Torque Absolute Max, 373
Transmission, 374, 383, 386
Transmission Gear High, 216
Transmission Gear Low, 216
Transmission Gear Ratio, 374, 386
Transmission protocol, 253
Transmission Protocol, type, 304–305
trapdata, 38
TriggC, 74
TriggCheckIO, 73
triggdata, 73
TriggEquip, 73
TriggInt, 73
TriggIO, 73
triggios, 73
triggiosdnum, 73
TriggJ, 74
TriggL, 74
TriggLIOs, 74
TriggRampAO, 74
TriggSpeed, 34
TriggStopProc, 34
triggstrgo, 73
Trusted, 253
TrustLevel, Task type, 283
TUNE_FRIC_LEV, 137
TUNE_FRIC_RAMP, 137
TuneServo, 137
Type, 253
Type, Task type, 283
Type, Transmission Protocol type, 304–305

U
UDP, 323
UdpUc, 316–317
Udp Unicast Communication, 316–317
Uncalibrated Control Master 0, 374
Unit_ready input signal, 385
UnpackRawBytes, 83
Upper Joint Bound, 372, 385
Use Activation Relay, 386
Use Linked Motor Process, 58
Use Process, 58
Use ramp time, 371
Username, 253

W
WaitSyncTask, 294
WaitSyncTask, instruction, 284
WaitUntil, 292
WAN端口, 341
WarmStart, 28
Wrist Move, 148
Write, 79
WriteAnyBin, 79
WriteBin, 79
WriteBlock, 306
WriteCfgData, 28
WriteRawBytes, 83
WriteStrBin, 79
WriteVar, 306
WZBoxDef, 211
WZCylDef, 211
WZDisable, 212
WZDOSet, 212
WZEnable, 212
WZFree, 212
WZHomeJointDef, 212
WZLimJointDef, 211
WZLimSup, 212
WZSphDef, 211
wzstationary, 211
wztemporary, 211

不
不允许停用, 204
不可打印字符, 265

与
与 (AND) , 95

丢
丢失消息, 273
丢失队列, 273

丧
丧失准确度, 118

中
中断, 47, 272, 292, 306, 308
中断功能, 37
中继器, 363

串
串行通道通信, 77

临
临时全局区域, 211

主
主动, 57

事
事件消息, 40
事件编号, 40
事件记录器, 262

二
二进制数据, 265
二进制通信, 78

交
交叉连接, 94

从

从动, 57

代

代码示例, 377

以

以太网, 251, 255

以太网链路, 342

任

任务, 281, 289

添加, 285

编辑程序, 285

设置, 285

任务优先级, 287

任务面板设定, 288

优

优先级, 287

传

传感器, 235, 302

配置, 303

传感器对象, 159

传感器系统, 203

传输协议, 253, 257

伺

伺服工具更换, 359

位

位功能, 18

位置事件, 72

位置警告, Machine Synchronization, 180

使

使用机器人校准, 114

使用连接中继器, 363

例

例程调用, 296

信

信号, 292, 296

俯

俯仰, 119

偏

偏移, 120

偏转, 119

停

停用任务, 289

停用监控, 249

元

元件

类型, 347

网络, 348

规约, 347

通道, 348

元素

字段, 350

性质, 350

成员, 349

枚举, 349

记录, 350

设定, 348

先

先决条件, 342

全

全局区域, 209

公

公用数据, 291

共

共享资源, 298

关

关节区域, 209

准

准确度的错误来源, 119

出

出厂证书, Absolute Accuracy, 124

函

函数

传感器接口, 306

多任务化, 284

高级RAPID语言, 45

切

切割形状, 153

切割面, 149

创

创建任务, 285

别

别名IO, 24

区

区域, 209

协

协议

串行通道, 304

以太网, 305

原

原始数据, 82

参

参数

准确度补偿, 125

发

发文消息, 343

发送消息, 355

受

受信任, 257

合

合成信号, 94–95

合规性错误, 119

同

同步多项任务, 294

同步移动, 367

名

名称, 204, 257

围

围笼的对准, 128

固

固定位置事件, 72
固定全局区域, 211
固定装置的对准情况, 129

在

在启动时激活, 204

坐

坐标系, 128

基

基于字符的通信, 78

声

声明, 291

外

外轴, 215, 240

套

套接字关闭, 266
套接字创建, 266
套接字发送, 266
套接字接受, 266
套接字接收, 266
套接字状态, 266
套接字监听, 266
套接字绑定, 266
套接字获取状态, 267
套接字装置, 266
套接字连接, 266

字

字段元素, 350
字符串终止, 265

安

安装, 375

对

对准, 128
对象队列, 159

导

导轨, 240

工

工具, 113, 360
工具更换校准, 368
工具校准, 132
工艺配套功能, 33

已

已关联信号, 204
已被抛弃的消息, 273
已记录的曲线, 194, 198
已记录的路径, 228

常

常数
传感器接口, 306

应

应用层协议, 253, 257

异

异步移动, 367

微

微调, 249
微调, 手动, 137
微调, 自动, 135

性

性质元素, 350

恢

恢复信号, 35
恢复路径, 220

意

意外断开连接, 375

成

成员元素, 349
成比例信号, 34

或

或 (OR) , 95

所

所收消息, 355

手

手动微调摩擦, 137
手动模式, Machine Synchronization, 180, 182

执

执行信号, 94–95

扭

扭矩, 242
扭矩从动件, 65
扭矩分配, 65

指

指令
传感器接口, 306
多任务化, 284
高级RAPID语言, 45

接

接受消息, 265

摩

摩擦等级微, 135
摩擦补偿, 134

效

效果限值, Machine Synchronization, 180

数

数字I / O信号, 94
数据, 271
数据交换, 340
数据变量
Electronically Linked Motors, 68
数据变量示例
Electronically Linked Motors, 70
数据就绪信号, 204
数据搜索功能, 21

数据类型

受支持, 344
多任务化, 284

文

文件管理, 86
文件结构, 86
文件通信, 77
文本表格文件, 40

断

断开连接, 375
断开连接时停用PTC监控, 362
断电功能, 31
断连停用, 362

旋

旋转变压器偏移校准, 122
旋转移动, 205

更

更换, 116
更换器, 360
更换工具, Machine Synchronization, 180
更改校准数据, 114
更新转数计数器, 376

最

最大先行距离, 203, 205
最大同步速度, 205
最大滞后距离, 205
最小同步速度, 205

服

服务, 375
服务器地址, 257
服务器路径, 257
服务程序, 61
服务连接, 341

未

未校准, 114, 376

本

本地路径, 257

机

机器人关节, 344
机器人的对准情况, 130
机械单元, 299
机械结构, 204
机械臂的更换, 117

枚

枚举元素, 349

标

标称速度, 203

校

校准从动轴, 62
校准工具, 113, 132
校准数据, 114
校准进程, 122
校正发生器, 235
校正自变数, 237

框

框架, 128
框架关系, 131

模

模块
传感器接口, 306
模拟信号, 47

横

横滚, 119

正

正信号, 204

每

每米计数量, 203

永

永久变量, 291

注

注塑机, 198

测

测量系统, 217

消

消息
发文, 343
已发送, 355
已收到, 355
消息合并, 265

液

液压式冲压机, 194

添

添加或更换参数, 169

激

激活Absolute Accuracy, 114
激活监控, 249

点

点动碰撞检测, 247
点动碰撞检测等级, 247

焊

焊枪头更换校准, 368
焊枪头磨损校准, 368

独

独立关节, 240
独立移动, 215
独立轴, 215

现

现场总线命令, 203

用

用户ID, 257
用户消息功能, 40

电

电机的更换, 116

监

监控等级, 244–245, 249

目

目录管理, 86

碰

碰撞, 241

禁

禁止激活, 365

移

移动, 241

程

程序

编辑, 285

程序指针, 45

等

等候若干任务, 294

等级, 347

类

类型, 257, 347

等级, 347

精

精确点, Machine Synchronization, 180

精细校准, 376

系

系统参数

传感器接口, 304–305

多任务化, 283

控制器主题, 343

运动主题, 343

配置功能, 27

系统资源, 298

绝

绝对准确度校准, 122

维

维护, 116

编

编程速度, Machine Synchronization, 180

编组ID, 257

编组I / O信号, 94

网

网络, 348

腕

腕的更换, 116

臂

臂的更换, 116

自

自动微调摩擦, 135

自动模式, 290

自变数名称, 45

虚

虚假目标点, 120

补

补偿, 120

补偿参数, 111

规

规约, 347

解

解除同步, 62

触

触发, 250

记

记录, 271

记录元素, 350

设

设定元素, 348

设置“碰撞检测”, 247

设置任务, 285

证

证书, Absolute Accuary, 124

误

误触发, 250

调

调度程序, 296

调节速度, 203

调试, 375

策略, 285

负

负载识别, 113

路

路径, 31

路径偏移, 235

路径分辨率, 204

路径恢复, 220

路径校正, 235

路径碰撞检测, 247

路径碰撞检测等级, 247

路径记录, 228

转

转数计数器未更新, 376

轮

轮询, 292

软

软中断例程, 272

软伺服器, 240

轴

轴, 215

轴重置, 215

载

载入校准数据, 114

输

输入信号, 363

运

运动命令, Machine Synchronization, 180

运动学错误, 119

运行模式, Machine Synchronization, 182

进

进程更新时间, 204

连

连接中继器, 363

选

选择任务, 289

通

通信客户端, 346

通信电缆

连接, 341

通信通道, 340

通道, 348

速

速度, 242

速度信号, 204

速度减少%按钮, Machine Synchronization, 180

速度警告, Machine Synchronization, 180

逻

逻辑AND, 96

逻辑OR, 96

逻辑轴, 362, 387

逻辑运算, 94

配

配置

Absolute Accuracy, 114

任务, 285

传感器, 303

配置“碰撞检测”, 247

配置功能, 27

配置参数, 125

配置文件, 345

配置示例, 352

重

重新连接一件伺服工具, 375

重置, 217

重置从动轴, 64

重置轴, 215

错

错误中断, 37

队

队列名称, 272

队列处理, 272

附

附加轴, 57, 366

降

降低定位准确度, 65

零

零速度信号, 204

验

验证, 123

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