

ROBOTICS

Application manual

Controller software OmniCore



Trace back information:
Workspace 24D version a9
Checked in 2024-12-18
Skribenta version 5.6.018

Application manual
Controller software OmniCore

RobotWare 7.17

Document ID: 3HAC066554-001

Revision: R

The information in this manual is subject to change without notice and should not be construed as a commitment by ABB. ABB assumes no responsibility for any errors that may appear in this manual.

Except as may be expressly stated anywhere in this manual, nothing herein shall be construed as any kind of guarantee or warranty by ABB for losses, damage to persons or property, fitness for a specific purpose or the like.

In no event shall ABB be liable for incidental or consequential damages arising from use of this manual and products described herein.

This manual and parts thereof must not be reproduced or copied without ABB's written permission.

Keep for future reference.

Additional copies of this manual may be obtained from ABB.

Original instructions.

© Copyright 2019-2024 ABB. All rights reserved.
Specifications subject to change without notice.

Table of contents

Overview of this manual	11
Open source and 3rd party components	14
1 Introduction to RobotWare	15
1.1 Products, classes, and options	15
1.2 RAPID language and programming environment	17
2 RobotWare-OS	19
2.1 Advanced RAPID	19
2.1.1 Introduction to Advanced RAPID	19
2.1.2 Bit functionality	20
2.1.2.1 Overview	20
2.1.2.2 RAPID components	21
2.1.2.3 Bit functionality example	22
2.1.3 Data search functionality	23
2.1.3.1 Overview	23
2.1.3.2 RAPID components	24
2.1.3.3 Data search functionality examples	25
2.1.4 Alias I/O signals	26
2.1.4.1 Overview	26
2.1.4.2 RAPID components	27
2.1.4.3 Alias I/O functionality example	28
2.1.5 Configuration functionality	29
2.1.5.1 Overview	29
2.1.5.2 RAPID components	30
2.1.5.3 Configuration functionality example	31
2.1.6 Power failure functionality	32
2.1.6.1 Overview	32
2.1.6.2 RAPID components and system parameters	33
2.1.6.3 Power failure functionality example	34
2.1.7 Process support functionality	35
2.1.7.1 Overview	35
2.1.7.2 RAPID components	36
2.1.7.3 Process support functionality examples	37
2.1.8 Interrupt functionality	39
2.1.8.1 Overview	39
2.1.8.2 RAPID components	40
2.1.8.3 Interrupt functionality examples	41
2.1.9 User message functionality	42
2.1.9.1 Overview	42
2.1.9.2 RAPID components	43
2.1.9.3 User message functionality examples	44
2.1.9.4 Text table files	46
2.1.10 RAPID support functionality	47
2.1.10.1 Overview	47
2.1.10.2 RAPID components	48
2.1.10.3 RAPID support functionality examples	49
2.2 Analog Signal Interrupt	50
2.2.1 Introduction to Analog Signal Interrupt	50
2.2.2 RAPID components	51
2.2.3 Code example	52
2.3 Connected Services	53
2.3.1 Overview	53
2.3.2 Connected Services connectivity	55
2.3.3 Connected Services registration	57
2.3.4 Summary of Connected Services paths in FlexPendant	59

2.3.5	Summary of Connected Services paths in RobotStudio	60
2.3.6	Configuration of system parameters	61
2.3.7	Configuring Connected Services using FlexPendant	63
2.3.7.1	Introduction	63
2.3.7.2	Enable or disable Connected Services using FlexPendant	64
2.3.7.3	Configure Connected Services based on connection type using FlexPendant	65
2.3.7.4	Configuration of public network using FlexPendant	68
2.3.7.5	Configure internet connection with proxy using FlexPendant	69
2.3.8	Configuring Connected Services using RobotStudio	70
2.3.8.1	Introduction	70
2.3.8.2	Enable or disable Connected Services using RobotStudio	71
2.3.8.3	Configure connected services based on connection type using RobotStudio	72
2.3.8.4	Configuration of public network using RobotStudio	75
2.3.8.5	Configure internet connection with proxy using RobotStudio	76
2.3.9	Connected Services information	77
2.3.10	Troubleshooting	87
2.3.10.1	Server connectivity troubleshooting	87
2.3.10.2	3G / Wi-Fi Connectivity troubleshooting	88
2.3.10.3	4G Connectivity troubleshooting	89
2.3.10.4	How to get Connected Services Embedded logs from the controller	90
2.3.10.5	Connected Services Embedded troubleshooting logs	91
2.3.11	Network topology scenarios	93
2.4	Cyclic bool	104
2.4.1	Cyclically evaluated logical conditions	104
2.4.2	Cyclic bool examples	107
2.4.3	System parameters	110
2.4.4	RAPID components	111
2.5	Device Command Interface	112
2.5.1	Introduction to Device Command Interface	112
2.5.2	RAPID components and system parameters	113
2.5.3	Code example	114
2.6	Electronically Linked Motors	116
2.6.1	Overview	116
2.6.2	Configuration	118
2.6.2.1	System parameters	118
2.6.2.2	Configuration example	120
2.6.3	Managing a follower axis	121
2.6.3.1	Using the service routine for a follower axis	121
2.6.3.2	Calibrate follower axis position	123
2.6.3.3	Reset follower axis	125
2.6.4	Tuning a torque follower	126
2.6.4.1	Description of torque follower	126
2.6.4.2	Using the service routine to tune a torque follower	128
2.6.5	Data setup	130
2.6.5.1	Set up data for the service routine	130
2.6.5.2	Example of data setup	132
2.7	File and I/O device handling	134
2.7.1	Introduction to file and I/O device handling	134
2.7.2	Binary and character based communication	135
2.7.2.1	Overview	135
2.7.2.2	RAPID components	136
2.7.2.3	Code examples	137
2.7.3	Raw data communication	139
2.7.3.1	Overview	139
2.7.3.2	RAPID components	140
2.7.3.3	Code examples	141

2.7.4	File and directory management	143
2.7.4.1	Overview	143
2.7.4.2	RAPID components	144
2.7.4.3	Code examples	145
2.8	Fixed Position Events	147
2.8.1	Overview	147
2.8.2	RAPID components and system parameters	148
2.8.3	Code examples	151
2.9	Logical Cross Connections	153
2.9.1	Introduction to Logical Cross Connections	153
2.9.2	Configuring Logical Cross Connections	154
2.9.3	Examples	155
2.9.4	Limitations	157
2.10	RAPID Message Queue	158
2.10.1	Introduction to RAPID Message Queue	158
2.10.2	RAPID Message Queue behavior	159
2.10.3	System parameters	163
2.10.4	RAPID components	164
2.10.5	Code examples	165
2.11	Socket Messaging	169
2.11.1	Introduction to Socket Messaging	169
2.11.2	Schematic picture of socket communication	170
2.11.3	Technical facts about Socket Messaging	171
2.11.4	RAPID components	172
2.11.5	Code examples for Socket Messaging	174
2.12	User logs	176
2.12.1	Introduction to User logs	176
3	Motion Performance	177
3.1	Absolute Accuracy [3101-x]	177
3.1.1	About Absolute Accuracy	177
3.1.2	Useful tools	179
3.1.3	Configuration	180
3.1.4	Maintenance	181
3.1.4.1	Maintenance that affect the accuracy	181
3.1.4.2	Loss of accuracy	183
3.1.5	Compensation theory	184
3.1.5.1	Error sources	184
3.1.5.2	Absolute Accuracy compensation	185
3.1.6	Preparation of Absolute Accuracy robot	187
3.1.6.1	ABB calibration process	187
3.1.6.2	Birth certificate	189
3.1.6.3	Compensation parameters	190
3.1.7	Cell alignment	191
3.1.7.1	Overview	191
3.1.7.2	Measure fixture alignment	192
3.1.7.3	Measure robot alignment	193
3.1.7.4	Frame relationships	194
3.1.7.5	Tool calibration	195
3.2	Ultra Accuracy [3101-10]	196
3.2.1	About Ultra Accuracy	196
3.2.2	Configuration	198
3.2.3	Compensation parameters	199
3.3	Advanced Robot Motion 3100-1	200
3.4	Advanced Shape Tuning [included in 3100-1]	201
3.4.1	About Advanced Shape Tuning	201
3.4.2	Automatic friction tuning	202
3.4.3	Manual friction tuning	204

Table of contents

3.4.4	System parameters	206
3.4.4.1	System parameters	206
3.4.4.2	Setting tuning system parameters	207
3.4.5	RAPID components	208
3.5	Motion Process Mode [included in 3100-1]	209
3.5.1	About Motion Process Mode	209
3.5.2	User-defined modes	211
3.5.3	General information about robot tuning	213
3.5.4	Additional information	216
3.6	Wrist Move [included in 3100-1]	217
3.6.1	Introduction to Wrist Move	217
3.6.2	Cut plane frame	219
3.6.3	RAPID components	221
3.6.4	RAPID code, examples	222
3.6.5	Troubleshooting	224
4	Motion Supervision	225
4.1	World Zones [3106-1]	225
4.1.1	Overview of World Zones	225
4.1.2	RAPID components	227
4.1.3	Code examples	229
4.2	Collision Detection [3107-1]	231
4.2.1	Overview	231
4.2.2	Limitations	233
4.2.3	What happens at a collision	234
4.2.4	Additional information	236
4.2.5	Configuration and programming facilities	237
4.2.5.1	System parameters	237
4.2.5.2	RAPID components	239
4.2.5.3	Signals	240
4.2.6	How to use Collision Detection	241
4.2.6.1	Set up system parameters	241
4.2.6.2	Adjust supervision from FlexPendant	242
4.2.6.3	Adjust supervision from RAPID program	243
4.2.6.4	How to avoid false triggering	244
4.3	Collision Avoidance [3150-1]	245
4.4	SafeMove Assistant	248
5	Motor Control	251
5.1	Independent Axis [3111-1]	251
5.1.1	Overview	251
5.1.2	System parameters	253
5.1.3	RAPID components	254
5.1.4	Code examples	255
6	RAPID Program Features	257
6.1	Path Recovery [3113-1]	257
6.1.1	Overview	257
6.1.2	RAPID components	258
6.1.3	Store current path	259
6.1.4	Path recorder	261
6.2	Multitasking [3114-1]	265
6.2.1	Introduction to Multitasking	265
6.2.2	System parameters	267
6.2.3	RAPID components	269
6.2.4	Communication between tasks	270
6.2.4.1	Persistent variables	270
6.2.4.2	Waiting for other tasks	272

6.2.4.3	Synchronizing between tasks	274
6.2.4.4	Using a dispatcher	276
6.2.5	Other programming issues	278
6.2.5.1	Share resource between tasks	278
6.2.5.2	Test if task controls mechanical unit	279
6.2.5.3	taskid	280
6.2.5.4	Avoid heavy loops	281
7	Communication	283
7.1	FTP&SFTP client [3116-1]	283
7.1.1	Introduction to FTP&SFTP client	283
7.2	NFS Client [3117-1]	285
7.2.1	Introduction to NFS Client	285
8	User Interaction Application	287
8.1	RobotStudio Connect [3119-1]	287
8.2	FlexPendant Base Apps	288
8.3	FlexPendant Independent Apps	289
9	Engineering tools	291
9.1	RobotWare Add-In	291
9.2	Path Corrections [3123-1]	292
9.2.1	Overview	292
9.2.2	RAPID components	294
9.2.3	Related RAPID functionality	295
9.2.4	Code example	296
9.3	Auto Acknowledge Input	297
10	Tool control options	299
10.1	Servo Tool Change [3110-1]	299
10.1.1	Overview	299
10.1.2	Requirements and limitations	300
10.1.3	Configuration	302
10.1.4	Connection relay	303
10.1.5	Tool change procedure	305
10.1.6	Jogging servo tools with activation disabled	306
10.2	Tool Control [3109-1]	307
10.2.1	Overview	307
10.2.2	Servo tool movements	308
10.2.3	Tip management	309
10.2.4	Supervision	311
10.2.5	RAPID components	312
10.2.6	System parameters	313
10.2.7	Commissioning and service	318
10.2.8	Mechanical unit calibrations	320
10.2.9	RAPID code example	321
Index		323

This page is intentionally left blank

Overview of this manual

About this manual

This manual explains the basics of when and how to use various RobotWare options and functions.

Usage

This manual can be used either as a reference to find out if an option is the right choice for solving a problem, or as a description of how to use an option. Detailed information regarding syntax for RAPID routines, and similar, is not described here, but can be found in the respective reference manual.

Who should read this manual?

This manual is intended for robot programmers.

Prerequisites

The reader should...

- be familiar with industrial robots and their terminology.
- be familiar with the RAPID programming language.
- be familiar with system parameters and how to configure them.

References



Tip

All documents can be found via myABB Business Portal, www.abb.com/myABB.

Reference	Document ID
<i>Product specification - OmniCore C line</i>	3HAC065034-001
<i>Product specification - OmniCore E line</i>	3HAC079823-001
<i>Product specification - OmniCore V line</i>	3HAC074671-001
<i>Operating manual - RobotStudio</i>	3HAC032104-001
<i>Operating manual - OmniCore</i>	3HAC065036-001
<i>Operating manual - Integrator's guide OmniCore</i>	3HAC065037-001
<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>	3HAC065038-001
<i>Technical reference manual - RAPID Overview</i>	3HAC065040-001
<i>Technical reference manual - System parameters</i>	3HAC065041-001
<i>Application manual - Additional axes</i>	3HAC082287-001
<i>Product manual - OmniCore C30 Type A</i>	3HAC089064-001
<i>Product manual - OmniCore C30</i>	3HAC060860-001
<i>Product manual - OmniCore C90XT</i>	3HAC073706-001
<i>Product manual - OmniCore E10</i>	3HAC079399-001

Continues on next page

Reference	Document ID
Product manual - OmniCore V250XT Type B	3HAC087112-001
Product manual - OmniCore V400XT	3HAC081697-001

Revisions

Revision	Description
A	Released with RobotWare 7.0.
B	Released with RobotWare 7.01. The following updates are made in this revision: <ul style="list-style-type: none"> • "Cyber security" replaced by "Cybersecurity" in entire manual. • Updated the section Connected Services on page 53.
C	Released with RobotWare 7.0.2. The following updates are made in this revision: <ul style="list-style-type: none"> • FlexPendant terminology updated in entire manual. • Updated the section Summary of Connected Services paths in FlexPendant on page 59.
D	Released with RobotWare 7.1. The following updates are made in this revision: <ul style="list-style-type: none"> • Updated the section Connected Services on page 53. • Added information about YuMi robots and Collision Detection, see Collision detection for YuMi robots on page 232. • Updated limitations for SFTP client, see Limitations on page 284.
E	Released with RobotWare 7.2. The following updates are made in this revision: <ul style="list-style-type: none"> • Updated the section Connected Services on page 53. • Information about the digital output <i>MotSupOn</i> updated in section Signals on page 240. • Section System parameters on page 163 updated with information about how to adjust the values of the attributes RMQ Max Message Size and RMQ Max No Of Messages. • Updated sections due to remote mounted disk/virtual root changes: Limitations on page 284 (FTP&SFTP client) and Limitations on page 285 (NFS Client).
F	Released with RobotWare 7.3. The following updates are made in this revision: <ul style="list-style-type: none"> • Updated the section Connected Services on page 53.
G	Released with RobotWare 7.4. The following updates are made in this revision: <ul style="list-style-type: none"> • Added the section Connected Services Embedded troubleshooting logs on page 91. • Updated the section Connected Services on page 53. • Updated information regarding UTF-8, in Raw data communication on page 139.
H	Released with RobotWare 7.5. <ul style="list-style-type: none"> • Updated limitation for Collision Avoidance [3150-1] on page 245.
J	Released with RobotWare 7.6. <ul style="list-style-type: none"> • Added the section Auto Acknowledge Input on page 297. • Updated limitation regarding lead-through, see Overview of World Zones on page 225. • Added the section SafeMove Assistant on page 248.

Continues on next page

Revision	Description
K	Released with RobotWare 7.10. <ul style="list-style-type: none"> • Added the option Servo Tool Change [3110-1] on page 299. • Minor corrections. • Removed statement that the instructions <code>StorePath</code> and <code>RestorePath</code> are included in RobotWare base. They require option <i>Path Recovery</i>. • Added information about deactivation/deactivation and trigger signals, see Collision Avoidance [3150-1] on page 245. • Minor corrections in Network topology scenarios on page 93. • Minor corrections in Auto Acknowledge Input on page 297.
L	Released with RobotWare 7.12. <ul style="list-style-type: none"> • Updated the section Connected Services information on page 77. • The function <i>Collision Avoidance</i> is now available for delta robots, see Limitations on page 246.
M	Released with RobotWare 7.13. <ul style="list-style-type: none"> • ABB Connected Services is the new name for ABB Ability Connected Services. • Added section Electronically Linked Motors on page 116. • Updated the section Connected Services on page 53.
N	Released with RobotWare 7.14. <ul style="list-style-type: none"> • Added section Tool Control [3109-1] on page 307.
P	Released with RobotWare 7.15. <ul style="list-style-type: none"> • Updated the server error details in the section Advanced page on page 81. • Updated the section Limitations on page 284 for FTP&SFTP Client.
Q	Released with RobotWare 7.16. <ul style="list-style-type: none"> • Added information about singularities and <i>Absolute Accuracy</i> in RobotStudio. • Added information about user configuration files for <i>Collision Avoidance</i>. • Added limitation in <i>Independent Axis</i> regarding tool control.
R	Released with RobotWare 7.17. <ul style="list-style-type: none"> • Added section Ultra Accuracy [3101-10] on page 196. • Added section RAPID language and programming environment on page 17. • Minor corrections.

Open source and 3rd party components

Open source and 3rd party components

ABB products use software provided by third parties, including open source software. The following copyright statements and licenses apply to various components that are distributed inside the ABB software. Each ABB product does not necessarily use all of the listed third party software components. Licensee must fully agree and comply with these license terms or the user is not entitled to use the product. Start using the ABB software means accepting also referred license terms. The third party license terms apply only to the respective software to which the license pertains, and the third party license terms do not apply to ABB products. With regard to programs provided under the GNU general public license and the GNU lesser general public license licensor will provide licensee on demand, a machine-readable copy of the corresponding source code. This offer is valid for a period of three years after delivery of the product.

ABB software is licensed under the ABB end user license agreement, which is provided separately.

RobotWare

For RobotWare, there is license information in the folder `\licenses` in the RobotWare distribution package.

OpenSSL

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)

This product includes cryptographic software written by Eric Young (ey@cryptsoft.com).

This product includes software written by Tim Hudson (tjh@cryptsoft.com).

CTM

For OleOS, the Linux based operating system used on the conveyor tracking module (CTM), a list of copyright statements and licenses is available in the file `/etc/licenses.txt` located on the CTM board and accessible via the console port or by downloading the file over SFTP.

For the CTM application, a list of copyright statements and licenses is available in the file `/opt/ABB.com/ctm/licenses.txt` located on the CTM board and accessible via the console port or by downloading the file over SFTP.

1 Introduction to RobotWare


1.1 Products, classes, and options

Software products

RobotWare is a family of software products from ABB Robotics. The products are designed to make you more productive and lower your cost of owning and operating a robot. ABB Robotics has invested many years into the development of these products and they represent knowledge and experience based on several thousands of robot installations.

Product classes

Within the RobotWare family, there are different classes of products:

Product classes	Description
RobotWare-OS	<p>This is the operating system of the robot. RobotWare-OS provides all the necessary features for fundamental robot programming and operation. It is an inherent part of the robot, but can be provided separately for upgrading purposes.</p> <p>For a description of RobotWare-OS, see the product specification for the robot controller.</p>
RobotWare options	<p>These products are options that run on top of RobotWare-OS. They are intended for robot users that need additional functionality for motion control, communication, system engineering, or applications.</p> <p> Note</p> <p>Not all RobotWare options are described in this manual. Some options are more comprehensive and are therefore described in separate manuals.</p>
Process application options	<p>These are extensive packages for specific process application like spot welding, arc welding, and dispensing. They are primarily designed to improve the process result and to simplify installation and programming of the application.</p> <p>The process application options are all described in separate manuals.</p>
RobotWare Add-ins	<p>A RobotWare Add-in is a self-contained package that extends the functionality of the robot system.</p> <p>Some software products from ABB Robotics are delivered as Add-ins. For example track motion IRBT, positioner IRBP, and stand alone controller. For more information see the product specification for the robot controller.</p> <p>The purpose of RobotWare Add-ins is also that a robot program developer outside of ABB can create options for the ABB robot systems, and sell the options to their customers. For more information on creating RobotWare Add-ins, contact your local ABB Robotics representative at www.abb.com/contacts.</p>

Continues on next page

1 Introduction to RobotWare

1.1 Products, classes, and options

Continued

Option groups

For OmniCore, the RobotWare options have been gathered in groups, depending on the customer benefit. The goal is to make it easier to understand the customer value of the options. However, all options are purchased individually. The groups are as follows:

Option groups	Description
Motion performance	Options that optimize the performance of your robot.
Motion coordination	Options that make your robot coordinated with external equipment or other robots.
Motion Events	Options that supervises the position of the robot.
Motion functions	Options that controls the path of the robot.
Motion Supervision	Options that supervises the movement of the robot.
Communication	Options that make the robot communicate with other equipment. (External PCs etc.)
Engineering tools	Options for the advanced robot integrator.
Servo motor control	Options that make the robot controller operate external motors, independent of the robot.



Note

Not all RobotWare options are described in this manual. Some options are more comprehensive and are therefore described in separate manuals.

1.2 RAPID language and programming environment

General

RAPID is the primary programming language used for ABB Robotics, designed to facilitate the control and automation of industrial robots. It is a high-level language that is both powerful and user-friendly, making it accessible for both novice and experienced programmers. Its syntax and structure are designed to be intuitive, reducing the learning curve for new users.

RAPID is suitable for a wide range of applications, from simple pick-and-place tasks to complex assembly operations. The language is designed to be reliable and robust, ensuring consistent performance in industrial environments.

Key features of RAPID

RAPID uses a structured text format similar to other programming languages like Python or C, which includes loops, conditionals, and variable handling. It excels in handling complex motion commands, allowing precise control over robot movements.

RAPID supports various data types and operations, enabling efficient data handling and processing. Users can create custom functions and procedures, enhancing the flexibility and adaptability of the programming environment.

It allows seamless communication with external devices and systems, making it ideal for integrated automation solutions.

Overall, RAPID is a versatile and powerful tool that enhances the capabilities of ABB robots, making automation more efficient and accessible.

Summary of the RAPID concept

- Hierarchical and modular program structure to support structured programming and reuse
- Routines can be *Functions* or *Procedures*
- Local or global data and routines
- Data typing, including structured and array data types
- User defined names on variables, routines, and I/O
- Extensive program flow control
- Arithmetic and logical expressions
- Interrupt handling
- Error handling
- User defined instructions (appear as an inherent part of the system)
- Backward handler (user definition of how a procedure should behave when stepping backwards)
- Many powerful built-in functions, for example mathematics and robot specific
- Unlimited language (no maximum number of variables etc., only memory limited). Built-in RAPID support in user interfaces, for example user defined pick lists, facilitate working with RAPID.
- Support for Unicode symbols in strings and comments

This page is intentionally left blank

2 RobotWare-OS

2.1 Advanced RAPID

2.1.1 Introduction to Advanced RAPID

Introduction to Advanced RAPID

The RobotWare base functionality *Advanced RAPID* is intended for robot programmers who develop applications that require advanced functionality.

Advanced RAPID includes many different types of functionality, which can be divided into these groups:

Functionality group	Description
Bit functionality	Bitwise operations on a byte.
Data search functionality	Search and get/set data objects (e.g. variables).
Alias I/O functionality	Give an I/O signal an optional alias name.
Configuration functionality	Get/set system parameters.
Power failure functionality	Restore signals after power failure.
Process support functionality	Useful when creating process applications.
Interrupt functionality	More interrupt functionality than included in RobotWare base functionality.
User message functionality	Error messages and other texts.
RAPID support functionality	Miscellaneous support for the programmer.

2 RobotWare-OS

2.1.2.1 Overview

2.1.2 Bit functionality

2.1.2.1 Overview

Purpose

The purpose of the bit functionality is to be able to make operations on a byte, seen as 8 digital bits. It is possible to get or set a single bit, or make logical operations on a byte. These operations are useful, for example, when handling a group of digital I/O signals.

What is included

Bit functionality includes:

- The data type `byte`.
- Instructions used set a bit value: `BitSet` and `BitClear`.
- Function used to get a bit value: `BitCheck`.
- Functions used to make logical operations on a byte: `BitAnd`, `BitOr`, `BitXOr`, `BitNeg`, `BitLSh`, and `BitRSh`.

2.1.2.2 RAPID components

Data types

This is a brief description of each data type used for the bit functionality. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
byte	The data type <code>byte</code> represent a decimal value between 0 and 255.

Instructions

This is a brief description of each instruction used for the bit functionality. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
BitSet	BitSet is used to set a specified bit to 1 in a defined byte data.
BitClear	BitClear is used to clear (set to 0) a specified bit in a defined byte data.

Functions

This is a brief description of each function used for the bit functionality. For more information, see the respective function in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
BitAnd	BitAnd is used to execute a logical bitwise AND operation on data types <code>byte</code> .
BitOr	BitOr is used to execute a logical bitwise OR operation on data types <code>byte</code> .
BitXOr	BitXOr (Bit eXclusive Or) is used to execute a logical bitwise XOR operation on data types <code>byte</code> .
BitNeg	BitNeg is used to execute a logical bitwise negation operation (one's complement) on data types <code>byte</code> .
BitLSh	BitLSh (Bit Left Shift) is used to execute a logical bitwise left shift operation on data types <code>byte</code> .
BitRSh	BitRSh (Bit Right Shift) is used to execute a logical bitwise right shift operation on data types <code>byte</code> .
BitCheck	BitCheck is used to check if a specified bit in a defined byte data is set to 1.



Tip

Even though not part of the option, the functions for conversion between a byte and a string, `StrToByte` and `ByteToStr`, are often used together with the bit functionality.

2 RobotWare-OS

2.1.2.3 Bit functionality example

2.1.2.3 Bit functionality example

Program code

```
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 Data search functionality

2.1.3.1 Overview

Purpose

The purpose of the data search functionality is to search and get/set values for data objects of a certain type.

Here are some examples of applications for the data search functionality:

- Setting a value to a variable, when the variable name is only available in a string.
 - List all variables of a certain type.
 - Set a new value for a set of similar variables with similar names.
-

What is included

Data search functionality includes:

- The data type `datapos`.
- Instructions used to find a set of data objects and get or set their values: `SetDataSearch`, `GetDataVal`, `SetDataVal`, and `SetAllDataVal`.
- A function for traversing the search result: `GetNextSym`.

2 RobotWare-OS

2.1.3.2 RAPID components

2.1.3.2 RAPID components

Data types

This is a brief description of each data type used for the data search functionality. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
datapos	datapos is the enclosing block to a data object (internal system data) retrieved with the function <code>GetNextSym</code> .

Instructions

This is a brief description of each instruction used for the data search functionality. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
<code>SetDataSearch</code>	<code>SetDataSearch</code> is used together with <code>GetNextSym</code> to retrieve data objects from the system.
<code>GetDataVal</code>	<code>GetDataVal</code> makes it possible to get a value from a data object that is specified with a string variable, or from a data object retrieved with <code>GetNextSym</code> .
<code>SetDataVal</code>	<code>SetDataVal</code> makes it possible to set a value for a data object that is specified with a string variable, or from a data object retrieved with <code>GetNextSym</code> .
<code>SetAllDataVal</code>	<code>SetAllDataVal</code> make it possible to set a new value to all data objects of a certain type that match the given grammar.

Functions

This is a brief description of each function used for the data search functionality. For more information, see the respective function in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
<code>GetNextSym</code>	<code>GetNextSym</code> (Get Next Symbol) is used together with <code>SetDataSearch</code> to retrieve data objects from the system.

2.1.3.3 Data search functionality examples

Set unknown variable

This is an example of how to set the value of a variable when the name of the variable is unknown when programming, and only provided in a string.

```
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;
```

Reset a range of variables

This is an example where all numeric variables starting with "my" is reset to 0.

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

List/set certain variables

In this example, all numeric variables in the module "mymod" starting with "my" are listed on the FlexPendant and then reset to 0.

```
VAR datapos block;
VAR string name;
VAR num valuevar;
VAR num zerovar:=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,zerovar;
ENDWHILE
```

2 RobotWare-OS

2.1.4.1 Overview

2.1.4 Alias I/O signals

2.1.4.1 Overview

Purpose

The Alias I/O functionality gives the programmer the ability to use any name on a signal and connect that name to a configured I/O signal.

This is useful when a RAPID program is reused between different systems. Instead of rewriting the code, using a signal name that exist on the new system, the signal name used in the program can be defined as an alias name.

What is included

Alias I/O functionality consists of the instruction `AliasIO`.

2.1.4.2 RAPID components

Data types

There are no RAPID data types for the Alias I/O functionality.

Instructions

This is a brief description of each instruction used for the Alias I/O functionality. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
AliasIO	AliasIO is used to define a signal of any type with an alias name, or to use signals in built-in task modules. The alias name is connected to a configured I/O signal. The instruction AliasIO must be run before any use of the actual signal.

Functions

There are no RAPID functions for the Alias I/O functionality.

2 RobotWare-OS

2.1.4.3 Alias I/O functionality example

2.1.4.3 Alias I/O functionality example

Assign alias name to signal

This example shows how to define the digital output signal `alias_do` to be connected to the configured digital output I/O signal `config_do`.

The routine `prog_start` is connected to the START event.

This will ensure that "alias_do" can be used in the RAPID code even though there is no configured signal with that name.

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

2.1.5 Configuration functionality

2.1.5.1 Overview

Purpose

The configuration functionality gives the programmer access to the system parameters at run time. The parameter values can be read and edited. The controller can be restarted in order for the new parameter values to take effect.

What is included

Configuration functionality includes the instructions: `ReadCfgData`, `WriteCfgData`, and `WarmStart`.

2 RobotWare-OS

2.1.5.2 RAPID components

2.1.5.2 RAPID components

Data types

There are no RAPID data types for the configuration functionality.

Instructions

This is a brief description of each instruction used for the configuration functionality. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
ReadCfgData	ReadCfgData is used to read one attribute of a named system parameter (configuration data).
WriteCfgData	WriteCfgData is used to write one attribute of a named system parameter (configuration data).
WarmStart	WarmStart is used to restart the controller at run time. This is useful after changing system parameters with the instruction WriteCfgData.

Functions

There are no RAPID functions for the configuration functionality.

2.1.5.3 Configuration functionality example

Configure system parameters

This is an example where the system parameter *cal_offset* for rob1_1 is read, increased by 0.2 mm and then written back. To make this change take effect, the controller is restarted.

```
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 RobotWare-OS

2.1.6.1 Overview

2.1.6 Power failure functionality

2.1.6.1 Overview

Purpose

If the robot was in the middle of a path movement when the power fail occurred, some extra actions may need to be taken when the robot motion is resumed. The power failure functionality helps you detect if the power fail occurred during a path movement.



Note

For more information see the type *Signal Safe Level*, which belongs to the topic *I/O System*, in *Technical reference manual - System parameters*.

What is included

The power failure functionality includes a function that checks for interrupted path:
`PFRestart`

2.1.6.2 RAPID components and system parameters

Data types

There are no RAPID data types in the power failure functionality.

Instructions

There are no RAPID instructions in the power failure functionality.

Functions

This is a brief description of each function in the power failure functionality. For more information, see the respective function in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
PFRestart	PFRestart (Power Failure Restart) is used to check if the path was interrupted at power failure. If so it might be necessary to make some specific actions. The function checks the path on current level, base level or on interrupt level.

System parameters

There are no system parameters in the power failure functionality. However, regardless of whether you have any options installed, you can use the parameter *Store signal at power fail*.

For more information, see *Technical reference manual - System parameters*.

2 RobotWare-OS

2.1.6.3 Power failure functionality example

2.1.6.3 Power failure functionality example

Test for interrupted path

When resuming work after a power failure, this example tests if the power failure occurred during a path (i.e. when the robot was moving).

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

2.1.7 Process support functionality

2.1.7.1 Overview

Purpose

Process support functionality provides some RAPID instructions that can be useful when creating process applications. Examples of its use are:

- Analog output signals, used in continuous process application, can be set to be proportional to the robot TCP speed.
- A continuous process application that is stopped with program stop or emergency stop can be continued from where it stopped.

What is included

The process support functionality includes:

- The data type `restartdata`.
- Instruction for setting analog output signal: `TriggSpeed`.
- Instructions used in connection with restart: `TriggStopProc` and `StepBwdPath`.

Limitations

The instruction `TriggSpeed` can only be used if you have the base functionality *Fixed Position Events*.

2 RobotWare-OS

2.1.7.2 RAPID components

2.1.7.2 RAPID components

Data types

This is a brief description of each data type used for the process support functionality. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
restartdata	restartdata can contain the pre- and post-values of specified I/O signals (process signals) at the stop sequence of the robot movements. restartdata, together with the instruction TriggStopProc is used to preserve data for the restart after program stop or emergency stop of self-developed process instructions.

Instructions

This is a brief description of each instruction used for the process support functionality. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
TriggSpeed	TriggSpeed is used to define the setting of an analog output to a value proportional to the TCP speed. TriggSpeed can only be used together with the option Fixed Position Events.
TriggStopProc	TriggStopProc is used to store the pre- and post-values of all used process signals. TriggStopProc and the data type restartdata are used to preserve data for the restart after program stop or emergency stop of self-developed process instructions.
StepBwdPath	StepBwdPath is used to move the TCP backwards on the robot path from a RESTART event routine.

Functions

There are no RAPID functions for the process support functionality.

2.1.7.3 Process support functionality examples

Signal proportional to speed

In this example, the analog output signal that controls the amount of glue is set to be proportional to the speed.

Any speed dip by the robot is time compensated in such a way that the analog output signal `glue_ao` is affected 0.04 s before the TCP speed dip occurs. If overflow of the calculated logical analog output value in `glue_ao`, the digital output signal `glue_err` is set.

```
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;
```



Tip

Note that it is also possible to create self-developed process instructions with `TriggSpeed` using the `NOSTEPIN` routine concept.

Resume signals after stop

In this example, an output signal resumes its value after a program stop or emergency stop.

The procedure `supervise` is defined as a `POWER ON` event routine and `resume_signals` as a `RESTART` event routine.

```
PERS restartdata myproc_data :=
  [FALSE,FALSE,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 \D01:=do_close_gun, do_close_gun;
```

Continues on next page

2 RobotWare-OS

2.1.7.3 Process support functionality examples

Continued

```
ENDPROC

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

Move TCP backwards

In this example, the TCP is moved backwards 30 mm in 1 second, along the same path as before the restart.

The procedure `move_backward` is defined as a RESTART event routine.

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

2.1.8 Interrupt functionality

2.1.8.1 Overview

Purpose

The interrupt functionality in Advanced RAPID has some extra features, in addition to the interrupt features always included in RAPID. For more information on the basic interrupt functionality, see *Technical reference manual - RAPID Overview*.

Here are some examples of interrupt applications that Advanced RAPID facilitates:

- Generate an interrupt when a persistent variable change value.
- Generate an interrupt when an error occurs, and find out more about the error.

What is included

The interrupt functionality in Advanced RAPID includes:

- **Data types for error interrupts:** `trapdata`, `errdomain`, and `errtype`.
- **Instructions for generating interrupts:** `IPers` and `IError`.
- **Instructions for finding out more about an error interrupt:** `GetTrapData` and `ReadErrData`.

2 RobotWare-OS

2.1.8.2 RAPID components

2.1.8.2 RAPID components

Data types

This is a brief description of each data type in the interrupt functionality. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
trapdata	trapdata represents internal information related to the interrupt that caused the current trap routine to be executed.
errdomain	errdomain is used to specify an error domain. Depending on the nature of the error, it is logged in different domains.
errtype	errtype is used to specify an error type (error, warning, state change).

Instructions

This is a brief description of each instruction in the interrupt functionality. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
IPers	IPers (Interrupt Persistent) is used to order an interrupt to be generated each time the value of a persistent variable is changed.
IError	IError (Interrupt Errors) is used to order an interrupt to be generated each time an error occurs.
GetTrapData	GetTrapData is used in trap routines generated by the instruction IError. GetTrapData obtains all information about the interrupt that caused the trap routine to be executed.
ReadErrData	ReadErrData is used in trap routines generated by the instruction IError. ReadErrData read the information obtained by GetTrapData.
ErrRaise	ErrRaise is used to create an error in the program and the call the error handler of the routine. ErrRaise can also be used in the error handler to propagate the current error to the error handler of the calling routine.

Functions

There are no RAPID functions for the interrupt functionality.

2.1.8.3 Interrupt functionality examples

Interrupt when persistent variable changes

In this example, a trap routine is called when the value of the persistent variable counter changes.

```

VAR intnum int1;
PERS num counter := 0;

PROC main()
  CONNECT int1 WITH iroutine1;
  IPers counter, int1;
  ...
  counter := counter + 1;
  ...
  Idelete int1;
ENDPROC

TRAP iroutine1
  TPWrite "Current value of counter = " \Num:=counter;
ENDTRAP

```

Error interrupt

In this example, a trap routine is called when an error occurs. The trap routine determines the error domain and the error number and communicates them via output signals.

```

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 RobotWare-OS

2.1.9.1 Overview

2.1.9 User message functionality

2.1.9.1 Overview

Purpose

The user message functionality is used to set up event numbers and facilitate the handling of event messages and other texts to be presented in the user interface.

Here are some examples of applications:

- Get user messages from a text table file, which simplifies updates and translations.
- Add system error number to be used as error recovery constants in RAISE instructions and for test in ERROR handlers.

What is included

The user message functionality includes:

- Text table operating instruction `TextTabInstall`.
- Text table operating functions: `TextTabFreeToUse`, `TextTabGet`, and `TextGet`.
- Instruction for error number handling: `BookErrNo`.

2.1.9.2 RAPID components

Data types

There are no RAPID data types for the user message functionality.

Instructions

This is a brief description of each instruction used for the user message functionality. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
BookErrNo	BookErrNo is used to define a new RAPID system error number.
TextTabInstall	TextTabInstall is used to install a text table in the system.

Functions

This is a brief description of each function used for the user message functionality. For more information, see the respective function in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
TextTabFreeToUse	TextTabFreeToUse is used to test whether the text table name is free to use (not already installed in the system).
TextTabGet	TextTabGet is used to get the text table number of a user defined text table.
TextGet	TextGet is used to get a text string from the system text tables.

2.1.9.3 User message functionality examples

Book error number

This example shows how to add a new error number.

```
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
```

Error message from text table file

This example shows how to get user messages from a text table file.

There is a text table named `text_table_name` in a file named `HOME:/language/en/text_file.xml`. This table contains error messages in english.

The procedure `install_text` is executed at event **POWER ON**. The first time it is executed, the text table file `text_file.xml` is installed. The next time it is executed, the function `TextTabFreeToUse` returns **FALSE** and the installation is not repeated.

The table is then used for getting user interface messages.

```
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
```

Continues on next page

```
ErrWrite TextGet(text_res_no, 1), TextGet(text_res_no, 2);
```

2 RobotWare-OS

2.1.9.4 Text table files

2.1.9.4 Text table files

Overview

A text table is stored in an XML file (each file can contain one table in one language). This table can contain any number of text strings with encoding UTF-8 or ISO-8859-1.

Explanation of the text table file

This is a description of the XML tags and arguments used in the text table file.

Tag	Argument	Description
Resource		Represents a text table. A file can only contain one instance of Resource.
	Name	The name of the text table. Used by the RAPID instruction <code>TextTabGet</code> .
	Language	Language code for the language of the text strings. The file installed with the RAPID instruction <code>TextTabInstall</code> is used for all languages. To use more than one language, install one file per language using a unique file path name and a unique Resource name.
Text		Represents a text string.
	Name	The number of the text string in the table.
Value		The text string to be used.
Comment		Comments about the text string and its usage.

Example of text table file

```
<?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 support functionality

2.1.10.1 Overview

Purpose

The RAPID support functionality consists of miscellaneous routines that might be helpful for an advanced robot programmer.

Here are some examples of applications:

- Activate a new tool, work object or payload.
 - Find out what an argument is called outside the current routine.
 - Test if the program pointer has been moved during the last program stop.
-

What is included

RAPID support functionality includes:

- Instruction for activating specified system data: `SetSysData`.
- Function that gets original data object name: `ArgName`.
- Function for information about program pointer movement:
`IsStopStateEvent`.

2 RobotWare-OS

2.1.10.2 RAPID components

2.1.10.2 RAPID components

Data types

There are no data types for RAPID support functionality.

Instructions

This is a brief description of each instruction used for RAPID support functionality. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
SetSysData	SetSysData activates (or changes the current active) tool, work object, or payload for the robot.

Functions

This is a brief description of each function used for RAPID support functionality. For more information, see the respective function in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
ArgName	ArgName is used to get the name of the original data object for the current argument or the current data.
IsStopStateEvent	IsStopStateEvent returns information about the movement of the program pointer.

2.1.10.3 RAPID support functionality examples

Activate tool

This is an example of how to activate a known tool:

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

This is an example of how to activate a tool when the name of the tool is only available in a string:

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

Get argument name

In this example, the original name of par1 is fetched. The output will be "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
```

Test if program pointer has been moved

This example tests if the program pointer was moved during the last program stop.

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

2 RobotWare-OS

2.2.1 Introduction to Analog Signal Interrupt

2.2 Analog Signal Interrupt

2.2.1 Introduction to Analog Signal Interrupt

Purpose

The purpose of Analog Signal Interrupt is to supervise an analog signal and generate an interrupt when a specified value is reached.

Analog Signal Interrupt is faster, easier to implement, and require less computer capacity than polling methods.

Here are some examples of applications:

- Save cycle time with better timing (start robot movement exactly when a signal reach the specified value, instead of waiting for polling).
- Show warning or error messages if a signal value is outside its allowed range.
- Stop the robot if a signal value reaches a dangerous level.

What is included

The RobotWare base functionality Analog Signal Interrupt gives you access to the instructions:

- `ISignalAI`
- `ISignalAO`

Basic approach

This is the general approach for using Analog Signal Interrupt. For a more detailed example of how this is done, see [Code example on page 52](#).

- 1 Create a trap routine.
- 2 Connect the trap routine using the instruction `CONNECT`.
- 3 Define the interrupt conditions with the instruction `ISignalAI` or `ISignalAO`.

Limitations

Analog signals can only be used if you have an industrial network option (for example DeviceNet).

2.2.2 RAPID components

Data types

Analog Signal Interrupt includes no data types.

Instructions

This is a brief description of each instruction in Analog Signal Interrupt. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
ISignalAI	<p>Defines the values of an analog input signal, for which an interrupt routine shall be called.</p> <p>An interrupt can be set to occur when the signal value is above or below a specified value, or inside or outside a specified range. It can also be specified if the interrupt shall occur once or repeatedly.</p>
ISignalAO	<p>Defines the values of an analog output signal, for which an interrupt routine shall be called.</p> <p>An interrupt can be set to occur when the signal value is above or below a specified value, or inside or outside a specified range. It can also be specified if the interrupt shall occur once or repeatedly.</p>

Functions

Analog Signal Interrupt includes no RAPID functions.

2.2.3 Code example

Temperature surveillance

In this example a temperature sensor is connected to the signal `ail`.

An interrupt routine with a warning is set to execute every time the temperature rises 0.5 degrees in the range 120-130 degrees. Another trap routine, stopping the robot, is set to execute as soon as the temperature rise above 130 degrees.

```
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 Connected Services

2.3.1 Overview

**Note**

ABB Connected Services is the new name for the functionality previously known as ABB Ability. During a period of time, both names will appear in and on our products.

Description

Connected Services is a functionality for ABB robot controllers to connect to ABB Connected Services Cloud by using 3G/4G, Wi-Fi, or wired connectivity. Connected Services collects the service information from the controller.

Purpose

The primary purpose of Connected Services is to collect service information from the controller.

These service information will be available through MyRobot, Connected Services portal, or pushed locally.

What is included

The RobotWare base functionality Connected Services gives you access to:

- a Connected Services agent software to manage the connectivity and the service data collection.
- system parameters used to enable and configure the connectivity.
- status and information pages.
- dedicated event logs for key events of Connected Services.
- connectivity through Connected Services Gateway.
- connectivity through the public port.

Prerequisites

The Connected Services function requires that the controller is included in a service agreement with ABB. Contact your local ABB office to create a service agreement with Connected Services and get access to MyRobot website to perform the registration after the connection.

**Note**

MyRobot is the ABB website which gives access to the service information of a robot controller under a service agreement.

Continues on next page

2 RobotWare-OS

2.3.1 Overview

Continued

Basic workflow

Connected Services is available natively as a plug and connect solution in RobotWare. The setup concept is to:

- 1 Provides internet connectivity to the robot controller.
- 2 Configure connected services and startup the connection.
- 3 Register the controller through MyRobot registration page.

Once Connected Services is connected and registered, the data collection will run transparently in the background.

Limitations

The controller identification is done using the controller serial number and must match the serial number defined in the service agreement.

Power On Connect

Using a Connected Services Gateway 3G or 4G will provide automatic connectivity after Power On of the controller without any configuration.

Production Registration

ABB will securely pre register Connected Services during production process to avoid the manual registration.

The manual registration is still available when needed.

2.3.2 Connected Services connectivity

Connected Services connection concept

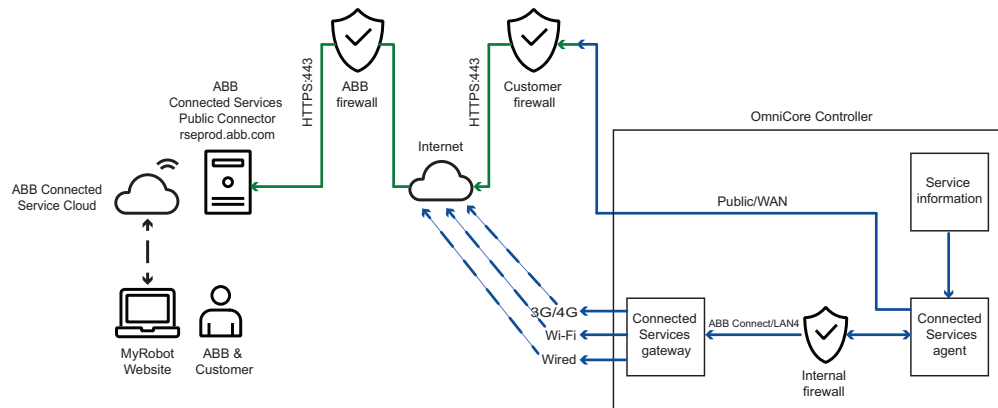
The concept of Connected Services is that a virtual Software Agent is implemented inside the controller and it communicates securely with the ABB Connected Services Cloud through internet.



Note

The connectivity of the controller through the public network requires a Firewall provided by the customer.

The communication is secured and encrypted using HTTPS (secure HTTP). The communication is possible only from the controller to ABB Cloud to keep the customer network isolated from any external Internet access. The following figure describe these concepts.



xx190000977

Disabling Connected Services

A robot controller can be delivered with Connected Services disabled by default, by selecting the option 3013-99 *No Connected Services* when ordering.

If the option 3013-99 is selected when ordering the controller, then the following apply:

- Connected Services embedded software is disabled.
- No Connected Services Gateway is installed.
- Detection of Connected Services Gateway is disabled.

A controller ordered with option 3013-99 can be retrofitted with Connected Services with the following procedure.

- 1 Configure the software, see [Enable or disable Connected Services using FlexPendant on page 64](#) or [Enable or disable Connected Services using RobotStudio on page 71](#).
- 2 Order a Connected Services Gateway, see spare parts in the product manual for the robot controller.

Continues on next page

2 RobotWare-OS

2.3.2 Connected Services connectivity

Continued

- 3 Enable the detection of the gateway, by configuring the *Connected Services Gateway* system parameters, see *Technical reference manual - System parameters*.
 - Enable 3G connection
 - Enable Wi-Fi connection
 - State (for wired connection)

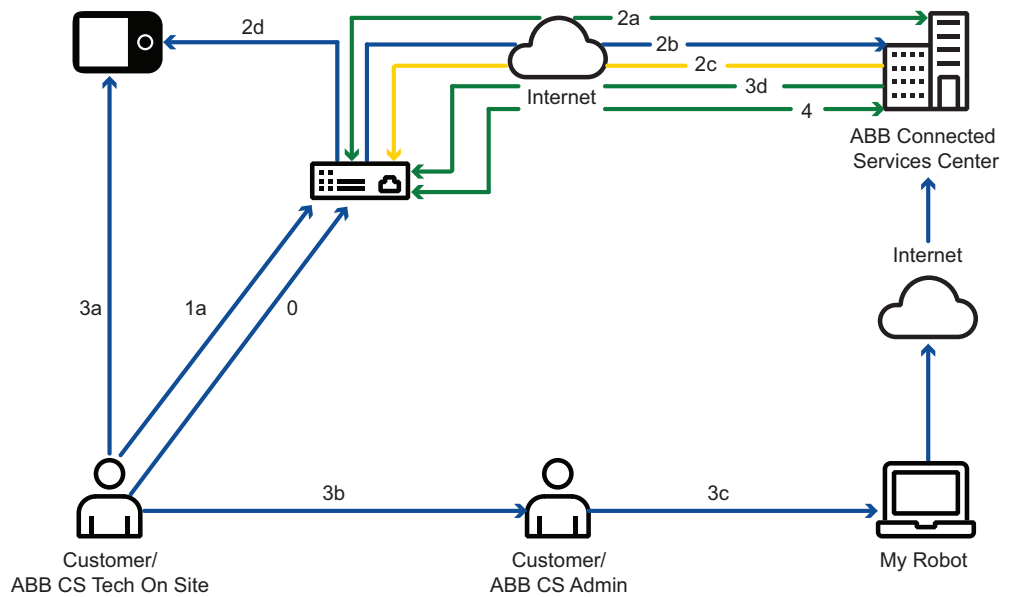
2.3.3 Connected Services registration

Connected Services startup

The Connected Services startup is based on the following steps:

- (0) Connected Services preparation
- (1) Connected Services configuration
- (2) Connected Services connectivity
- (3) Connected Services registration
- (4) Connected Services connected and registered

When these steps are done, the software agent is securely connected and identified with a client certificate. The following figure describes these concepts:



xx1500003226

Step	Description
0	Check controller S/N and internet connectivity
1a	Enable CSE and set up connectivity configuration
2a	CS connectivity in place
2b	Low poll for registration
2c	Registration not trusted (get reg code)
2d	Display registration code
3a	Get registration code
3b	Give controller S/N and registration code
3c	Select controller S/N in SA and register with registration code
3d	Registration trusted (client certificate)
4	Connected and registered secure CS session

Continues on next page

2 RobotWare-OS

2.3.3 Connected Services registration

Continued

Connected Services preparation

- 1 Verify that the service agreement for this controller is available with ABB.
- 2 Verify the controller serial number with the serial number found in the controller cabinet.
- 3 Verify and provide Internet connectivity to the robot controller.

Connected Services configuration

- 1 Configure the connectivity parameters based on connection type, see [Configuring Connected Services using FlexPendant on page 63](#).

Connected Services connectivity

- 1 The software agent connects to ABB Connected Services Cloud.
- 2 An initial registration process starts based on the selected polling rate.
- 3 The initial registration is incomplete and not yet fully trusted.
- 4 A registration code is received to finalize the trust relation.
- 5 The registration code is made available on the Connected Services registration page.

Connected Services registration

- 1 The customer/ABB on site provides the controller serial number and registration code to the Connected Services administrator for registration.
- 2 The Connected Services administrator validates this registration code in *MyRobot/Registration* for ABB Connected Services on its service agreement.
- 3 The registration trust starts and implements a client certificate in the controller.

Connected Services connected and registered

- 1 The controller is connected, registered, and identified as in the service agreement.
- 2 The connection is trusted with a client certificate.
- 3 Connected Services is now actively running on the robot controller.



Note

In case of Power On Connect with 3G or 4G Connected Services Gateway and production registration, all these process is done automatically when the controller is powered.

2.3.4 Summary of Connected Services paths in FlexPendant

Configuration

In Flexpendant the Connected Services configuration are available in:

- CS Gateway 3G: **Settings > ABB Connected Services > 3G Connection**
- CS Gateway WiFi: **Settings > ABB Connected Services > WiFi Connection**
- CS Gateway Wired: **Settings > ABB Connected Services > Wired Connection**



Note

Connected Services Gateway 4G is configured through Wired Connection.

Status

In FlexPendant the Connected Services status are available in:

- Network: **Settings > ABB Connected Services > Network Status**
- CS Gateway: **Settings > ABB Connected Services > Connectivity Status**
- Connected Services summary: **QuickSet > Connected Services**
- Connected Services details: **Settings > ABB Connected Services > Connected Services Status**
- Reset Connected Services: **Operate > Service Routine**

Logs

In Flexpendant the Connected Services logs are available in:

- Connection Logs (3G/WiFi): **Settings > Backup and Recovery > Connection Logs**
- Details Logs: **Settings > Backup and Recovery > System Diagnostic**

2 RobotWare-OS

2.3.5 Summary of Connected Services paths in RobotStudio

2.3.5 Summary of Connected Services paths in RobotStudio

Configuration

In RobotStudio the Connected Services configuration are available in:

- **CS Gateway 3G: Controller > Configuration > Communication > CS Gateway 3G**
- **CS Gateway WiFi: Controller > Configuration > Communication > CS Gateway WiFi**
- **CS Gateway Wired: Controller > Configuration > Communication > CS Gateway Wired**



Note

Connected Services Gateway 4G is configured through Wired Connection.

Status

In RobotStudio the Connected Services status are available in:

- **Connected Services Summary/Details: Controller > Properties > Device Browser > Software Resources > Connected Services**
- **Reset Connected Services: Not implemented**

Logs

In RobotStudio the Connected Services logs are available in:

- **Connection Logs (3G/4G/Wi-Fi): Not implemented**
- **Details Logs: Controller > Properties > Save Diagnostic**



2.3.6 Configuration of system parameters

Introduction

This section provides a brief description of system parameters used for the Connected Services. For more information see *Technical reference manual - System parameters*.

Connected Services connection

The following parameters belong to the topic *Communication*, and the type *Connected Services*. For more information, see the respective parameter in *Technical reference manual - System parameters*.


Parameter	Description
Enabled	Enables or disables the Connected Services connection between the controller and the server.
Connection Type	Indicates if the communication is done on ABB Connect (ABB Connected Services Gateway solution), Public, or Custom network.  Note If the connection type is configured as Public or Custom , then enable Connected Services on Firewall Settings. For more details, see the section Firewall settings in <i>Operating manual - Integrator's guide OmniCore</i> .
Internet Gateway IP	Defines the internet gateway IP of the connected network when the connection type is private. This is valid for the Connection Type Custom. Forces the Internet Gateway to use for connectivity.
Internet DNS IP	Defines the internet DNS IP of the connected network. This is valid for the Connection Type Custom. Forces the DNS to use for connectivity.  Note The DNS can be set to blank if: <ul style="list-style-type: none"> • the proxy is defined. • the DNS resolution is done with the proxy. • the proxy is provided as an IP address.
Proxy Used	Defines if a proxy is used to access the internet and its name/address, port, and authentication.
Proxy Auth	Defines the proxy authentication type. Basic will use HTTP basic authentication including user and password. None will not use any.
Server Polling	Defines the frequency of polling for specific synchronization with the sever. The available values are <code>slow</code> and <code>fast</code> . For details about the behavior of events for server polling, see Description of behavior of events for server polling on page 85 .
Debug Mode	Enables extensive logging for debugging the issues.
Trace Level	Defines the level of logging if Debug Mode is enabled.

Continues on next page

2 RobotWare-OS

2.3.6 Configuration of system parameters

Continued

Parameter	Description
Connected Services Mode	<p>Defines the compatibility for different robot controller's data format, cloud solution, and specific features. Following are the available modes:</p> <ul style="list-style-type: none">• Omnicore data collection. <p> Note</p> <p>By default, the Omnicore data collection mode is enabled.</p>

WAN configuration (Public connectivity)

The WAN IP/Mask/Gateway configuration is done in RobotStudio or on the FlexPendant. The WAN Ethernet port configuration can give access to the Internet on the controller without using the Connected Services Gateway. The port is defined by its IP, Mask, and possible Gateway. For details about Public (WAN) configuration, see the section Configuring Connected Service Gateway using FlexPendant in *Operating manual - Integrator's guide OmniCore*.



Note

A firewall must be installed by the customer, if the Public/WAN port is connected to Internet.

DNS configuration (Public connectivity)

These parameters belong to the topic *Communication* and the type *DNS Client*. A DNS server need to be defined to resolve the name of the ABB Connected Services connector (rseprod.abb.com) to its IP address, if Public/WAN port is used for internet connectivity. For more details, see *Type DNS Client* in *Technical reference manual - System parameters*.



Note

For quick testing, use a temporary DNS defined as 8.8.8.8 (Google DNS), then switch to customer recommended DNS server IP.

2.3.7 Configuring Connected Services using FlexPendant

2.3.7.1 Introduction

Overview

This section explains how Connected Services is configured using the controller FlexPendant based on the available internet connectivity. Internet connectivity can be provided in multiple ways.

- Connected Services Gateway module (3G, 4G, Wi-Fi, or wired)
- Direct internet connection on customer Public (WAN) network
- Direct internet connection on custom network

2.3.7.2 Enable or disable Connected Services using FlexPendant

Enabling or disabling Connected Services

This section provides information about enabling or disabling Connected Services using FlexPendant.



Note

Connected Services is enabled by default.

Use the following procedure to enable or disable Connected Services on the FlexPendant:

- 1 On the start screen, tap **Settings**, and then select **ABB Connected Services** from the menu.
- 2 Tap **Connected Services** on the left pane.
- 3 In the **Enable Connected Services** field tap and select the value **Yes** or **No**.
- 4 Tap **Apply**.
The **Restart** confirmation message is displayed.
- 5 Tap **OK**.
The controller is restarted and Connected Services is enabled or disabled based on the selection.

2.3.7.3 Configure Connected Services based on connection type using FlexPendant

Overview

Connected Services can be configured in the following ways depending on the available connection type:

- ABB Connect
- Public
- Custom

Configuring the connection type ABB Connect

Connected Services is configured with the connection type **ABB Connect** when the ABB Connected Services Gateway solution is connected.



Note

The connection type **ABB Connect** is enabled by default.

Use the following procedure to configure the connection type **ABB Connect** using FlexPendant:

- 1 On the start screen, tap **Settings**, and then select **ABB Connected Services** from the menu.
- 2 Tap **Connected Services** on the left pane.
The configuration parameters for Connected Services is displayed.
- 3 In the **Connection Type** list, tap and select **ABB Connect**.



Note

The ABB Connected Services network can be configured based on the available CS Gateway (3G, 4G, Wi-Fi, or Wired). For details, see *Operating manual - Integrator's guide OmniCore*.

- 4 Tap **Apply**.
The **Restart** confirmation message is displayed.
- 5 Tap **Yes**.
The controller is restarted.
Connected Services starts communicating to the server based on the configuration.
Check the connectivity status or event logs. For more details, see [Connected Services information on page 77](#).

Continues on next page

2 RobotWare-OS

2.3.7.3 Configure Connected Services based on connection type using FlexPendant

Continued

Configuring the connection type **Public**

Connected Services can be configured with the connection type *Public* when the communication is done on customer *Public* (WAN) network using the IP, the default gateway and DNS received through DHCP or statically configured.



Note

The *Public* network and DNS can be configured statically or automatically (via DHCP). For more details, see [Configuration of public network using FlexPendant on page 68](#).



Note

If the connection type is configured as **Public**, then enable Connected Services on Firewall Settings. For more details, see the section **Firewall settings** in *Operating manual - Integrator's guide OmniCore*.

Use the following procedure to configure the public network using FlexPendant:

- 1 On the start screen, tap **Settings**, and then select **ABB Connected Services** from the menu.
- 2 Tap **Connected Services**.
The configuration parameters for connected services are displayed.
- 3 In the **Connection Type** list, tap and select **Public**.
- 4 Tap **Apply**.
The **Restart** confirmation message is displayed.
- 5 Tap **Yes**.
The controller is restarted.

Connected Services starts communicating to the server based on the configuration.

Check the connectivity status or event logs. For more details, see [Connected Services information on page 77](#).

Configuring the connection type **Custom**

Connected Services can be configured with the connection type *Custom* when the controller has to specify a default Gateway and DNS available on the network.



Note

If the connection type is configured as **Custom**, then enable Connected Services on Firewall Settings. For more details, see the section **Firewall settings** in *Operating manual - Integrator's guide OmniCore*.

Use the following procedure to configure the connection type *Custom* using FlexPendant:

- 1 On the start screen, tap **Settings**, and then select **ABB Connected Services** from the menu.

Continues on next page

2.3.7.3 Configure Connected Services based on connection type using FlexPendant *Continued*

- 2 Tap **Connected Services** on the left pane.

The configuration parameters for connected services is displayed.

- 3 In the **Connection Type** list, tap and select **Custom**.

- 4 In the **Internet Gateway IP** field, type the IP address of internet gateway.

- 5 In the **Internet DNS IP** field, type the IP address of Internet DNS.

- 6 Tap **Apply**.

The **Restart** confirmation message is displayed.

- 7 Click **OK**.

The controller is restarted.

Connected Services starts communicating to the server based on the configuration.

Check the connectivity status or event logs. For more details, see [Connected Services information on page 77](#).

2.3.7.4 Configuration of public network using FlexPendant

Configuring IP and DNS Statically

Use the following procedure to statically configure IP and DNS using FlexPendant:

- 1 On the start screen, tap **Settings**, and then select **Network** from the menu.
- 2 Tap **Public Network** on the left pane.
- 3 Select the option **Use the following IP Address** or **Use the following DNS server addresses**.
- 4 Enter the values in **IP address**, **Subnet mask**, **Default gateway**, **Preferred DNS server**, and **Alternate DNS server** fields.
- 5 Tap **Apply**.

The IP and DNS are configured statically.

Configuring IP and DNS Automatically

Use the following procedure to automatically configure IP and DNS using FlexPendant:

- 1 On the start screen, tap **Settings**, and then select **Network** from the menu.
- 2 Tap **Public Network** on the left pane.
- 3 Select the option **Automatically get an IP Address** or **Automatically get DNS server addresses**.

The IP address and DNS server address is uploaded automatically.

- 4 Tap **Apply**.

The IP and DNS are configured automatic.

2.3.7.5 Configure internet connection with proxy using FlexPendant

Procedure

The following procedure provides information about configuring the Connected Services from the FlexPendant when there is Internet connection with proxy.

- 1 On the start screen, tap **Settings**, and then select **ABB Connected Services** from the menu.
- 2 Tap **Connected Services**.
The configuration parameters for Connected Services are displayed.
- 3 In the **Proxy Used** field, change the value to **Yes**.
The proxy parameters are displayed.
- 4 In the **Proxy Name** field, type a name for the proxy.
- 5 In the **Proxy Port** field, type the proxy port number.
- 6 In the **Proxy Auth** field, select **Basic** for basic authentication or select **None** for no authentication from the drop-down list.



Note

Define the proxy user name and password for the basic authentication. Even if a proxy is used, it is mandatory to define a DNS for name resolution.



Note

If your RobotWare version is between RW 7.8.1 and RW 7.10, it is mandatory to define a username and password in proxy configuration, even if it is not used.

- 7 Tap **Apply** and restart the controller to take effect of the changes.

2.3.8 Configuring Connected Services using RobotStudio

2.3.8.1 Introduction

Overview

This section explains how the Connected Services is configured using RobotStudio with the controller based on the available internet connectivity. Internet connectivity can be provided in multiple ways.

- Connected Services Gateway Module (3G, 4G, Wi-Fi, or wired)
- Direct internet connection on Customer Public (WAN) network
- Direct internet connection on custom network

2.3.8.2 Enable or disable Connected Services using RobotStudio

Enabling or disabling connected services

This section provides information about enabling or disabling Connected Services using RobotStudio.



Note

Connected services is enabled by default.

Use the following procedure to manage the enabling or disabling of the connected services feature:

- 1 Add controller in RobotStudio.
- 2 Click controller **Configuration**.
- 3 Right-click on **Communication** and select **Configuration Editor**.
The **Configuration Editor** is displayed.
- 4 Select **Connected Services**.
The configuration parameters for connected services is displayed.
- 5 Right-click on any field and select **Edit Connected Services(s)**.
The **Instance Editor** is displayed.
- 6 In the **Enabled** field select the value **Yes** or **No**.
- 7 Click **OK**.
- 8 Restart the controller.
The connected services is enabled or disabled based on the selection.

2.3.8.3 Configure connected services based on connection type using RobotStudio

Overview

Connected services can be configured in the following three ways depending on the available connection type:

- **ABB Connect**
- **Public**
- **Custom**

Configuring the connection type ABB Connect

Connected services is configured with the connection type **ABB Connect** when the ABB Connected Services Gateway solution is connected.



Note

The connection type **ABB Connect** is enabled by default.

Use the following procedure to configure the connection type **ABB Connect** using RobotStudio:

- 1 In the **Controller** tab, click **Configuration**.
- 2 Right-click on **Communication** and select **Configuration Editor**.
The **Configuration Editor** is displayed.
- 3 Select **Connected Services**.
The configuration parameters for connected services is displayed.
- 4 Right-click on any field and select **Edit Connected Services(s)**.
The **Instance Editor** is displayed.
- 5 In the **Connection Type** field, select the value **ABB Connect**.



Note

Network can be configured based on the available CS Gateway (3G, 4G, Wi-Fi, or Wired). For details, see *Operating manual - Integrator's guide OmniCore*.

- 6 Click **OK**.
- 7 Restart the controller.
The connected services start communicating to the server based on the configuration.
Check the connectivity status in the device browser. For more details, see [Connected Services information on page 77](#).
Also refer to the event logs generated. For more details, see *Technical reference manual - Event logs for RobotWare 7*.

Continues on next page

2.3.8.3 Configure connected services based on connection type using RobotStudio

Continued

Configuring the connection type Public

Connected services can be configured with the connection type **Public** when the communication is done on customer WAN network and when there is a default gateway and DNS received.



Note

Public network and DNS can be configured statically or automatic (through DHCP). For more details, see [Configuration of public network using RobotStudio on page 75](#).



Note

If the connection type is configured as **Public**, then enable Connected Services on Firewall Settings. For more details, see the section **Firewall settings** in *Operating manual - Integrator's guide OmniCore*.

Use the following procedure to configure the the connection type **Public** using RobotStudio:

- 1 In the **Controller** tab, click **Configuration**.
- 2 Right-click on **Communication** and select **Configuration Editor**.
The **Configuration Editor** is displayed.
- 3 Click **Connected Services**.
The configuration parameters for connected services is displayed.
- 4 Right-click on any field and select **Edit Connected Services(s)**.
The **Instance Editor** is displayed.
- 5 In the **Connection Type** field, select the value **Public Network**.
- 6 Click **OK**.
- 7 Restart the controller.

The connected services start communicating to the server based on the configuration.

Check the connectivity status in the device browser. For more details, see [Connected Services information on page 77](#).

Also refer to the event logs generated. For more details, see *Technical reference manual - Event logs for RobotWare 7*.

Configuring the connection type Custom

Connected services can be configured with the connection type **Custom** when the controller has to specify a default gateway and DNS available on the network.



Note

If the connection type is configured as **Custom**, then enable Connected Services on Firewall Settings. For more details, see the section **Firewall settings** in *Operating manual - Integrator's guide OmniCore*.

Continues on next page

Continued

Use the following procedure to configure the connection type **Custom** using RobotStudio:

- 1 In the **Controller** tab, click **Configuration**.
- 2 Right-click on **Communication** and select **Configuration Editor**.
The **Configuration Editor** is displayed.
- 3 Click **Connected Services**.
The configuration parameters for connected services is displayed.
- 4 Right-click on any field and select **Edit Connected Services(s)**.
The **Instance Editor** is displayed.
- 5 In the **Connection Type** field, select the value **Private Network**.
- 6 In the **Internet Gateway IP** field, type the IP address of internet gateway.
- 7 In the **Internet DNS IP** field, type the IP address of Internet DNS.
- 8 Click **OK**.
- 9 Restart the controller.

The connected services start communicating to the server based on the configuration.

Check the connectivity status in the device browser. For more details, see [Connected Services information on page 77](#).

Also refer to the event logs generated. For more details, see *Technical reference manual - Event logs for RobotWare 7*.

2.3.8.4 Configuration of public network using RobotStudio

Configuring IP Statically

Use the following procedure to configure statically IP using RobotStudio:

- 1 Right-click on the controller and select **Properties > Network Settings**.
The **Network settings** window is displayed.
- 2 Select the option **Use following IP address**.
- 3 Enter the values in **IP address, Subnet mask, Default gateway** fields.
- 4 Click **OK** and restart the controller
The IP is configured.

Configuring DNS Statically

The following procedure provides information about configuring the Connected Services from RobotStudio when there is direct internet connection with statically DNS.

- 1 In the **Controller** tab, click **Configuration**.
- 2 Right-click on **Communication** and select **Configuration Editor**.
The **Configuration Editor** is displayed.
- 3 Click **DNS Client**.
The configuration parameters for DNS Client is displayed.
- 4 Right-click on any field and select **Edit DNS Client(s)**.
The **Instance Editor** is displayed.
- 5 In the **Enabled** field change the value to **Yes**.
- 6 In the **1st Name Server** field type the server IP.
- 7 Click **OK** and restart the controller for the changes to take effect.

Configuring IP Automatic (DHCP)

Use the following procedure to configure IP automatic using RobotStudio:

- 1 Right-click on the controller and select **Properties > Network Settings**.
The **Network settings** window is displayed.
- 2 Select the option **Obtain an IP address automatically**.
The IP address is uploaded automatically.
- 3 Click **OK** and restart the controller
The IP and DNS shall be received automatically.



Note

It is still possible to define manual DNS with automatic IP. If there is conflict between manual and automatic DNS, manual DNS has priority.

2.3.8.5 Configure internet connection with proxy using RobotStudio

Procedure

The following procedure provides information about configuring the Connected Services from the RobotStudio when there is internet connection with proxy.

- 1 In the **Controller** tab, click **Configuration**.
- 2 Right-click on **Communication** and select **Configuration Editor**.
The **Configuration Editor** is displayed.
- 3 Select **Connected Services**.
The configuration parameters for connected services is displayed.
- 4 Right-click on any field and select **Edit Connected Services(s)**.
The **Instance Editor** is displayed.
- 5 In the **Proxy Used** field, select the value **Yes**.
The proxy parameters are displayed.
- 6 In the **Proxy Name** field, type a name for the proxy.
- 7 In the **Proxy Port** field, type the proxy port number.
- 8 In the **Proxy Auth** field, from the drop-down list, select *Basic* for basic authentication or select *None* for no authentication.



Note

Define the **Proxy User** and **Proxy Password** fields for the basic authentication. Even if a proxy is used, it is mandatory to define a DNS for name resolution.



Note

If your RobotWare version is between RW 7.8.1 and RW 7.10, it is mandatory to define a username and password in proxy configuration, even if is not used.

- 9 Click **OK**.
The controller is restarted and the changes are applied.

2.3.9 Connected Services information

Connected Services pages

Introduction

The Connected Services information pages are available in FlexPendant under **Settings > ABB Connected Services > Connected Services Status**. The following Connected Services Status information pages are available:

- **Overview**
- **Connectivity**
- **Registration**
- **Advanced**



Note

The Connected Services information pages are available in RobotStudio under **Controller Properties > Device Browser > Software resources > Communication > Connected Services**.



Note

The information on a page can be refreshed by changing the page or by pressing the **Refresh** button. The **Refresh** button also forces a connection with the server if the software agent is waiting (for example, wait for registration acknowledgement from MyRobot). This is useful in case of slow polling when Server Polling is set to Slow.

Overview page

The **Overview** page provides a summary of the Connected Services status and information. If the status is not active then the other pages provide more detailed information.

Field	Description	Possible values	Example
Property			
Enabled	Displays the value of the master configuration switch for turning the Connected Services on/off.	Yes/No	Yes
Status	Displays the current status to see whether there is a need to navigate to the Server connection page or Registration page.	For a description of values, see CSE status on page 83 .	Active
Serial number	Displays the identifier that is used to identify the controller in Connected Services.	Controller Serial number	12-45678
Robot OS Version	Displays the Robot OS Version that is sent to the server.	Robot OS Version number	RobotOS_1.00.0-379
Robot Control version	Displays the Robot Control version that is sent to the server.	Robot Control version name	RobotControl_7.0.0-405.Internal+405

Continues on next page

2 RobotWare-OS


2.3.9 Connected Services information

Continued


Field	Description	Possible values	Example
Service Agreement	To verify that the controller is associated to the expected service agreement.	"Name of the service agreement" "-"	SA_FR12_16
Customer name	To verify that the controller is associated to the expected service agreement.	"Customer Name of the service agreement" "-"	ABB Robotics
Country	To verify that the controller is associated to the expected service agreement.	"Country of the service agreement" "-"	France
ABB server	Displays the type of the server connected services is connected.	Robotics Cloud	Robotics Cloud

Connectivity page

The **Connectivity** page provides a summary of the Connected Services connectivity to the server.

Field	Description	Possible values	Example
Status	Displays the current status to see whether there is a need to navigate to the Connectivity page or Registration page.	For a description of values, see CSE status on page 83 .	Active
Connection Status	Displays the status of communication with the server and the type of error.	For a description of values, see CSE connection status on page 83 .	Connected
Last updated	Displays the relative time since the information on the Connectivity page has been generated.		"HH:MM:SS ago"
Hardware gateway	Displays the type of hardware gateway and connection IP.  Note 4G Gateway is displayed as DSQC1041 Wired.		DSQC 1039 3G Connected on IP: 192.168.126.1
Server name	Displays the name of the server that software agent is configured with.	"" Server name	Connected Services - rseprod.abb.com

Continues on next page


Field	Description	Possible values	Example
Server IP	<p>Displays the IP address of the server and the port number used for connection. The IP address is the result of DNS name resolution done by software agent.</p> <p> Note</p> <p>The IP address of resprod.abb.com is displayed only if DNS is resolved locally. If the system is unable to resolve the IP, value displayed will be "none".</p> <p>If proxy is configured and the DNS resolution is done by the proxy server then the value displayed will be "Proxy".</p>	"" Server IP	138.227.175.43 for rseprod.abb.com
Server certificate name	Displays the server certificate name information.	"" Server name Untrusted (Server)	Connected Services - rseprod.abb.com
Server certificate issuer	Displays the name of the server certificate issuer.	"" Issuer Untrusted (Issuer)	ABB issuing CA 6 DigiCert SHA2 Secure Server CA
Server certificate valid from	Displays the server certificate date.	"" Issuer Issued (Date)	Oct 02 08:07:12 2020 GMT
Server certificate valid until	Displays the server certificate date.	"" Issuer Expired (Date)	Nov 21 07:09:28 2021 GMT
Client certificate device	Displays the name of the client certificate device.		07-000036
Client Certificate issuer	Displays the name of the client certificate issuer.		Remote-Service- PROD-Issuing-CA-1
Client Certificate valid from	Displays from which date onwards the client certificate is valid.		Mar 15 05:38:41 2019 GMT
Client Certificate valid until	Displays till which date the client certificate is valid.		Mar 15 05:38:41 2020 GMT
Client Certificate serial number	Displays the serial number of the client certificate.		15E37B17000100 002D0B
Internet IP	Displays the IP which is used to connect to internet.		106.197.204.16

Continues on next page

2 RobotWare-OS

2.3.9 Connected Services information

Continued

Field	Description	Possible values	Example
Controller time	Displays the controller date and time details.  Note It is important to set the correct time in the controller as this is needed for the certificate process.		16-01-08 13:52:33
Connection type	Displays the type of network connection used.	Public ABB Connect Custom	
Public network information	Displays the network information for the Public port.	NoIP Static DHCP	DHCP: 10.140.198.55/ 255.255.255.0/ 10.140.198.1/ plugged
ABB Connect network information	Displays the network information for the ABB Connect port.	NoIP Static DHCP	Static: 192.168.126.2/ 255.255.255/ 0.0.0.0/ plugged
System DNS	Displays the DNS server information.		8.8.8.8:53
Gateway used	Displays the gateway used for creating the routes.		192.168.126.1
DNS used	Displays the DNS values that are currently used.		192.168.126.1:53
Route 1-8	Displays the routes created by Connected Services.		Route 1 : 13.79.129.11/32 => 192.168.126.1

Registration page

The **Registration** page provides a summary of the Connected Services registration.

Field	Description	Possible values	Example
Status	Displays the current status to see whether there is a need to navigate to the Server connection page or Registration page.	For a description of values, see CSE status on page 83 .	Active
Registration Status	Displays the registration status.	For a description of values, see CSE registration status on page 84 .	Register with code in MyRobot
Registration code	Displays the registration code. This code can be used to register using MyRobot.	"-" Code value	456735

Continues on next page

Advanced page

The **Advanced** page provides advanced information about the dialog between software agent and server.

Field	Description	Possible values	Example
Last HTTP message	Displays the last message sent.	Register CheckRegistered LogMessage GetMessage GetConfig SendGetSpecific-Code DownLoadFile AcknowledgeMessage BoxUpload GetRSEAgreementInformation SendDeviceInformation RequestClientCertificate RenewClientCertificate periodicDeviceUpdateInformation	GetMessage
Last HTTP message time	Displays the date and time when the last message was sent.		Sent 00:01:28 ago
Last HTTP error	Displays the HTTP error when the last message was sent and the message ID if 4XX.	Not Available - if no error Error HTTP XXX + Message	Not Available
Next message	Displays the next message to send and the date to send the message.		GetMessage in 70 seconds
Last command	Displays the last command received from server.	Not Available Reboot Reset Ping Diagnostic ...	Not Available
Restart counter	Displays the number of times the software Agent been auto-restarted. This is used to see if watchdog has restarted.	0-N If not Enabled, then display 0	2
Connector version	Displays the currently running connector version information.		1.0.0

Continues on next page

2 RobotWare-OS

2.3.9 Connected Services information

Continued

Field	Description	Possible values	Example
Data Collector status	Displays the status of the data collector.	Started Not Started None Downloaded Download Failed Failed Stopped Disabled	Started
Last registered	Displays the last registration date and time to server of controller.		
Last connected	Displays the last date and time on which the controller detects a successful connection with the server.		11-07-2020 08:33:35
Server polling	Displays the Server polling configuration. Server polling is related to the calculation of connection cost.	Fast Slow	Fast
Bytes sent/received	Displays the number of bytes sent/received.		17.0KB/24.17KB
Connected Services mode	Displays the status of the connected services mode.	For a description of values, see CSE mode on page 85 .	Active Mode
Server errors	Displays a count of the following servers errors: 1 Connection errors 2 Connection not available errors 3 Authentication errors 4 Request errors 5 Timeout errors 6 Proxy errors 7 Unknown errors 8 Server errors	0-N for each server error	5 0 1 0 1 0 0 4
Root certificate issuer	Displays the name of the root certificate issuer.		Baltimore CyberTrust Root DigiCert Global Root CA
Root certificate subject	Displays the name of the root certificate subject.		Baltimore CyberTrust Root DigiCert Global Root CA
Root certificate valid from	Displays from which date onwards the root certificate is valid.		May 12 18:46:00 2000 GMT May 12 18:46:00 2006 GMT
Root certificate valid until	Displays till which date the root certificate is valid.		May 12 18:46:00 2025 GMT May 12 18:46:00 2031 GMT

Continues on next page

Data collectors page

The Data Collectors are the components used to collect the data required by ABB Connected Services Cloud. The Data Collectors are updatable OTA (Over The Air) from ABB Cloud.

The **Data Collector** page provides information about the state and version details of different data collectors.

Field	Description	Possible values	Example
OmniCore Connected Services	Displays the status and version information of Connected Services data collector.	Disabled Version information	1.0.4

Description of values in Connected Services information

CSE status

The following table gives the information of CSE status:

Code	Value	Description
BASE_FAILED	Failed	Connected Services status is failed.
BASE_INITIALIZING	Initializing	Connected Services status is initializing.
BASE_ACTIVE	Active	Connected Services status is active.
BASE_CONNECT	Trying to connect	Connected Services status is trying to connect.
BASE_SHUTDOWN	Shutdown mode	Connected Services status is shutdown mode.
UNKNOWN_STATUS	Unknown	Connected Services status is unknown.
BASE_SLEEP	Sleep	Connected Services status is sleep mode.

CSE connection status

The following table gives the information of CSE connection status:

Code	Value	Description
SERVER_REQUEST_TIMEOUT_ERROR	Request timed out	Connection status request is timed out.
SERVER_CONNECTED	Connected	Connection status in connected.
SERVER_NETWORK_ERROR	Server not reachable	Connection status server is not reachable.
SERVER_AUTH_ERROR	Server not authenticated	Connection status server is not authenticated.
SERVER_CERT_ERROR	Server certification verification error	Connection status server is certification verification error.
SERVER_HTTP_ERROR	Server error (HTTP)	Connection status server is HTTP error.
SERVER_PROXY_AUTH_ERROR	Proxy Authentication Required	Connection status proxy authentication error is required.
SERVER_REQUEST_ERROR	Request error	Connection status is request error.
SERVER_PROXY_CONN_ERROR	Proxy Connection Error	Connection status proxy connection error.

Continues on next page

2 RobotWare-OS

2.3.9 Connected Services information

Continued

Code	Value	Description
SERVER_GATEWAY_DISABLED	CS Gateway disabled	Connection status Gateway is disabled.
SERVER_GATEWAY_WAITING	Identifying CS Gateway	Connection status Gateway is identifying.
SERVER_GATEWAY_WAITING_3G	Waiting 3G connectivity	Connection status Gateway is waiting for 3G connectivity.
SERVER_GATEWAY_WAITING_WIFI	Waiting Wi-Fi connectivity	Connection status Gateway is waiting for Wi-Fi connectivity.
SERVER_GATEWAY_CONNECTED	CS Gateway connected	Connection status Gateway is connected.
SERVER_LOOKUP_IN_PROGRESS	Host name lookup in progress	Connection status Gateway host name lookup in progress.
WAITING_GATEWAY_IP	Waiting for Gateway IP	Connection status Gateway is waiting for Gateway IP.
SERVER_DNS_RESOLUTION_FAIL	DNS resolution failed	Connection status Gateway has server DNS resolution failure.
SERVER_GW_PING_TIMEOUT	Gateway ping timeout	Connection status Gateway server ping is time out.
DNS_NOT_PINGABLE	DNS not pingable	Connection status Gateway has DNS that is not pingable.
AUTHN_ERROR	Authentication error	Connection status Gateway having authentication error.
RESTART_ERROR	Waiting to start after restart error	Connection status Gateway is waiting to start after restart error.
CS_BLOCKED	Blocked	CSE is displaying this message when CSE is not enabled in Public or Private network. By default CSE is enabled on Private network. For more details, see the parameter External Firewall Enabled in <i>Technical reference manual - System parameters</i> .
RESET_ERROR	Waiting to start after reset error	Connection status Gateway is waiting to start after reset error.
CSE_SLEEP_MODE	CSE in configured sleep mode	Connection status Gateway is in sleep mode.

CSE registration status

The following table gives the information of CSE registration status:

Code	Value	Description
REG_REGISTERED	Registered	Registration status is registered.
REG_IN_PROGRESS	Registration in progress	Registration status is in progress.
REG_REGISTER	Registration with code in MyRobot	Registration status is registered with code in MyRobot.
REG_DISABLED	Registration disabled	Registration status is disabled.
REG_FAILED	Failed	Registration status is failed.

Continues on next page

CSE mode

The following table gives the information of CSE mode:

Code	Value	Description
CS_MODE_BOOT	Boot Mode	Connected Services during initialization.
CS_MODE_REGISTRATION	Registration Mode	Connected Services in registration mode.
CS_MODE_ACTIVE	Active Mode	Connected Services after registration is successful.
CS_MODE_RESET	Reset Mode	Connected Services during reset of connected services.
CS_MODE_SLEEP	Sleep Mode	When there is a delay to perform operation.
CS_MODE_SHUTDOWN	Shutdown Mode	When suspend connected services until revoke.

Data collection status

The following table gives the information of data collection status:

Code	Value	Description
SPECIFIC_STATUS_NONE	None	Data collection status is none.
SPECIFIC_STATUS_DOWNLOADED	Download	Data collection status is downloaded.
SPECIFIC_STATUS_DOWNLOAD_FAILED	Download Failed	Data collection status downloaded failed.
SPECIFIC_STATUS_STARTED	Started	Data collection status is started.
SPECIFIC_STATUS_STOPPED	Stopped	Data collection status is stopped.
SPECIFIC_STATUS_FAILED	Failed	Data collection status is failed.

Description of behavior of events for server polling

The following table gives information about the behavior of events for server polling:

Events	Slow	Fast
Check Registration	30 min	10 min
Send HardWare Information (before registration) (Periodic polling is send only if there is any change.)	30 min	10 min
Send HardWare Information (after registration)	24 h	24 h
Send Alive message	50 min	50 min
Check for Module Update	8 h	4 h
Send Dynamic Information	4 h	4 h
Internal Pending Command	10 min	1 min
Restart after Unregistration	Restart watchdog (4 minutes)	

Continues on next page

2 RobotWare-OS

2.3.9 Connected Services information

Continued

Connected Services event logs

The software agent generates event logs in the central controller event log. Event logs are generated during starting, registering, unregistering, losing connectivity, and during other key events.

The events logs are in the range of 170XXX and are described with all the other controller event logs documentation. For more details, see *Technical reference manual - Event logs for RobotWare 7*.

Force a reset of the software agent

It is possible to reset the software agent. When you reset, the software agent erases all its internal information including the registration information, the data collector script, and all the locally stored service information. The configuration will not be reset, but a new registration is required to reactivate the connected services.

Use the following procedure to reset the software agent using FlexPendant:

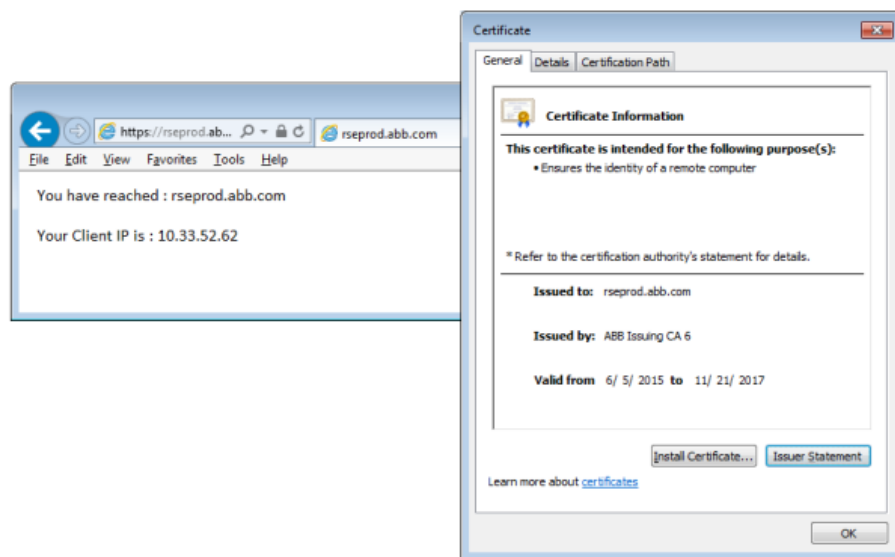
- 1 **Open Operate.**
- 2 **Tap Service Routines.**
- 3 **Tap Connected Services Reset.**
The **ConnectedServiceReset** window is displayed.
- 4 **Tap Yes.**
- 5 **Press the START hard button on the FlexPendant.**
A confirmation page is displayed with operator messages.
- 6 **Tap Reset.**
The software agent is reset.

2.3.10 Troubleshooting

2.3.10.1 Server connectivity troubleshooting

Overview

From your location it is possible to verify the connectivity from the controller to the Connected Services public connector server. This is done by connecting a PC (instead of the controller) with the same network configuration (WAN IP/Mask, DNS, Route), and open the path to the root of the Connected Services server (<https://rseprod.abb.com>) in a browser. The connectivity is validated if the DNS name has been resolved, the browser presents a page indicating the CS server, and secured with an ABB certificate as shown in the following figures.



xx1500003225

Connected Services Gateway

For more details, see the section ABB Connected Services configuration in *Operating manual - Integrator's guide OmniCore*.

Cybersecurity

For more details, see the section OmniCore Cybersecurity in *Operating manual - Integrator's guide OmniCore*.

Time accuracy

It is important to set up the time correctly in the controllers including Time Zone, either manually or with NTP. For more details, see the section **Changing date and time** in *Operating manual - OmniCore*.

2.3.10.2 3G / Wi-Fi Connectivity troubleshooting

Overview

This option is used to check the current connection state of the connectivity module for troubleshooting.



Note

Connection log is available only for Connected Services Gateway 3G and Wi-Fi (not for Wired).

Procedure

Use the following procedure to check the current connection state of the connectivity module:

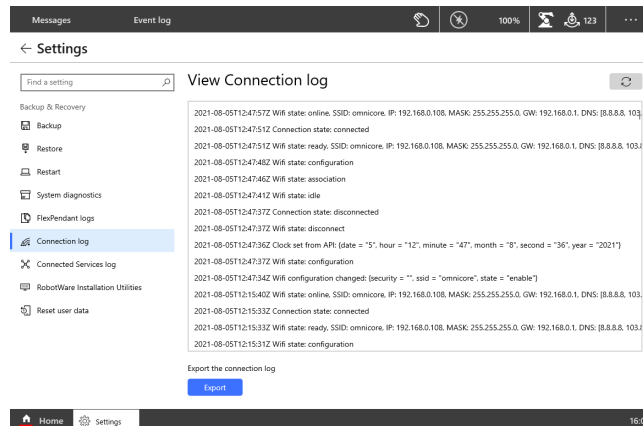
- 1 Open **Settings**.
- 2 Tap **Backup & Recovery**.
- 3 On the left sidebar tap **Connection log**.

The **Connection log** page is displayed and the logs are displayed on a window.



Note

Tap on the refresh button to update the logs.



xx190000976

- 4 Tap **Export**.
- 5 If required, in the **File Name** field edit the name of the file.
- 6 If required, to change the storage path, in the **Folder Name** field tap **Browse** and select the required path.
- 7 Tap **Create**.

The current connection state of the connectivity module is saved in the selected path.

2.3.10.3 4G Connectivity troubleshooting

Overview

It is possible to get some status information and logs from the Connected Services Gateway 4G. This is done by using an external PC connected on port Eth 1 of the 4G Gateway.

For more information about troubleshooting, see the product manual for the controller, listed in [References on page 11](#).

2.3.10.4 How to get Connected Services Embedded logs from the controller

Procedure

Use the following procedure to retrieve the CSE logs from the controller:

- 1 Open RobotStudio, click the **Controller** tab, and add the controller.



Note

For more details about adding the controller, see *Operating manual - RobotStudio*.

- 2 In the **Configuration** group, click **Properties** and select **Save diagnostics**.
The **Save As** window is displayed.
- 3 Click **Save**.
The file is saved in the selected location.
- 4 Send the full diagnostic file to ABB support for further processing.

2.3.10.5 Connected Services Embedded troubleshooting logs

Connected Services Embedded troubleshooting logs and description

Connected Services Embedded (CSE) sends some log messages to Connectivity Management Secure Server (CMSS). These log messages are categorized into two types, periodic logs and non-periodic logs. Periodic logs contain CSE health status. Non-periodic logs are sent when CSE enters a particular state like registered, Data collector started, and so on. These logs messages are helpful in troubleshooting CSE. The logs are accessible from internal ABB Connected Services support tool.

Connected Services Embedded base logs

The following table provides the list of Connected Services Embedded base logs and its description:


Log number Log name	Description
3000 BASE_MODE_UNKNOWN	Controller received unsupported reset command from external source.
3170 BASE_MODE_FAIL_UPDATE	Omnicores data collection update has been failed.
8000 GLOBAL_INIT_OK	Connected Services Embedded (CSE) is successfully registered with CMSS server.
8033 INFO_JAVA_SPECIFIC_LOAD_NOK	Connected Services Embedded (CSE) is unable to start Data collector.
8123 INFO_EXT_JAVARESET	Connected Services Embedded (CSE) received reset command from external source.

Continues on next page

2 RobotWare-OS

2.3.10.5 Connected Services Embedded troubleshooting logs

Continued

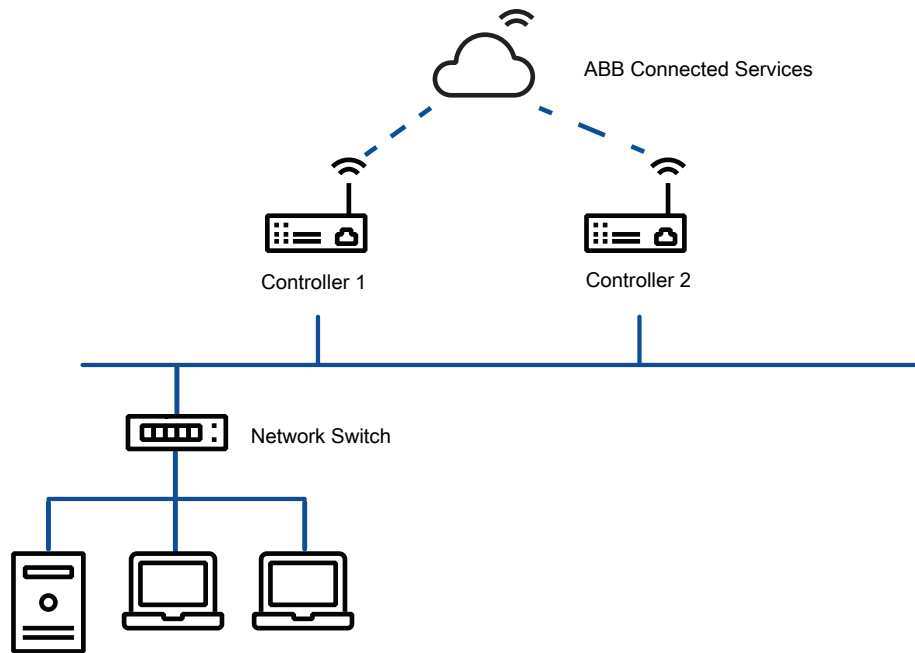
Log number Log name	Description
<p>8210 INFO_STAY_ALIVE</p> <p> Note</p> <p>INFO_STAY_ALIVE is a periodic log.</p>	<p>Connected Services Embedded (CSE) sends keep-alive messages to CMSS server at predefined intervals (50 minutes) to indicate connected service is alive. The alive message format is as below.</p> <p>Example:</p> <p>message_count Alive Bytes:xx/yy HTTP:p/q/r/s/t/u/v Mem:7069164 Run:746 RC:0 LHE:</p> <ul style="list-style-type: none"> • message_count: Current alive message count. • Bytes: Number of bytes of memory sent and received by controller. • HTTP: Different http errors occurred while controller communicating with CMSS server. <p>Type of errors:</p> <ul style="list-style-type: none"> - p: Connection error count. - q: Connection not available error count. - r: Authentication related error count. - s: Request error count. - t: Timeout error count. - u: Proxy error count. - v: Unknown error count. • Mem: Free memory available in bytes. • Run: Controller's uptime in seconds. • RC: Number of times CSE restarted since the last boot. • LHE: Information about last http error.
<p>8213 INFO_ALIVE_STARTED</p>	<p>Connected Services Embedded (CSE) will send keep alive message once CSE registered. The message format is same as 8210 INFO_STAY_ALIVE.</p>
<p>8214 INFO_ALIVE_ENDED</p>	<p>Connected Services Embedded (CSE) will send this alive message when Connected Services Embedded stopped. The message format is same as 8210 INFO_STAY_ALIVE.</p>
<p>8700 INFO_MODULE_UPDATE_OK</p>	<p>Connected Services Embedded (CSE) module updated successfully.</p>
<p>8701 INFO_MODULE_UPDATE_NOK</p>	<p>Connected Services Embedded (CSE) module update failed.</p>
<p>8702 INFO_MODULE_UPDATE_ERROR</p>	<p>Connected Services Embedded (CSE) module update is failed.</p>
<p>8801 INFO_S24_STARTED</p>	<p>Connected Services Embedded (CSE) data collector module start.</p>
<p>8803 INFO_S2301_STARTED</p>	<p>Connected Services Embedded (CSE) connector(S2301) connector started.</p>
<p>9003 INFO_SPECIFIC_STOP</p>	<p>Connected Services Embedded (CSE) data collector stopped.</p>

2.3.11 Network topology scenarios

Connection Type – Connected Services with 4G module

In the following scenario the Controller 1 and Controller 2 are installed and configured with ABB SIM. Refer to Connection Settings table in the following figure for detailed configuration. Based on this configuration Connected Services Gateway 4G module will connect to the network.

Connected Services is configured with ABB Connect Connection Type, which means all the communication to the ABB Cloud will pass and be routed through the Connected Services Gateway 4G module.



xx1900001182

4G connection settings for controller 1 and 2	Controller 1 (NW1)	Controller 2
State	Enabled	Enabled
IP Address	192.168.126.2	192.168.126.2
Subnet mask	255.255.255.0	255.255.255.0
Default gateway	192.168.126.1	192.168.126.1
Preferred DNS	192.168.126.1	192.168.126.1
Secondary DNS	0.0.0.0	0.0.0.0

Connected services settings	
State	Enabled
Connection type	ABB Connect
Proxy used	No
Server polling	Slow recommended
Connected services mode	Omnicores data collection

Continues on next page

2 RobotWare-OS

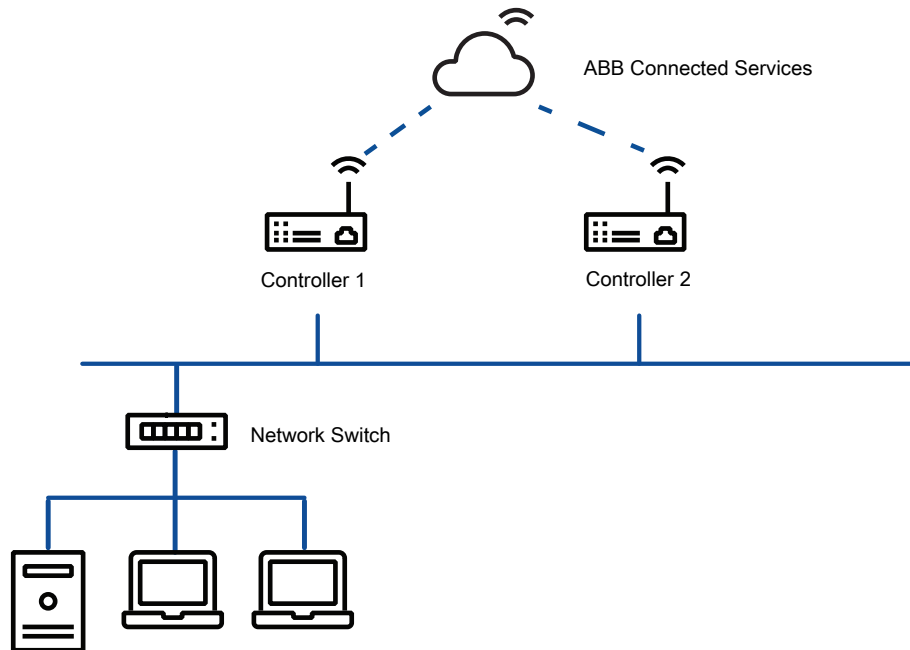
2.3.11 Network topology scenarios

Continued

Connection Type – Connected Services with 3G module

In the following scenario the Controller 1 and Controller 2 are installed and configured with ABB SIM. Refer to Connection Settings table in the following figure for detailed configuration. Based on this configuration Connected Services Gateway 3G module will connect to the network.

Connected Services is configured with ABB Connect Connection Type, which means all the communication to the ABB Cloud will pass and be routed through the Connected Services Gateway 3G module.



xx1900001182

3G connection settings for controller 1 and 2	
State	Enabled
APN (Access point)	abbrobotics.com
Operator	Automatic
Band	Automatic
Auth	Automatic
Roaming	Enabled
Idle	0
Delay	0
Connected services settings	
State	Enabled
Connection type	ABB Connect
Proxy used	No
Server polling	Slow or Fast
Connected services mode	Omnicores data collection

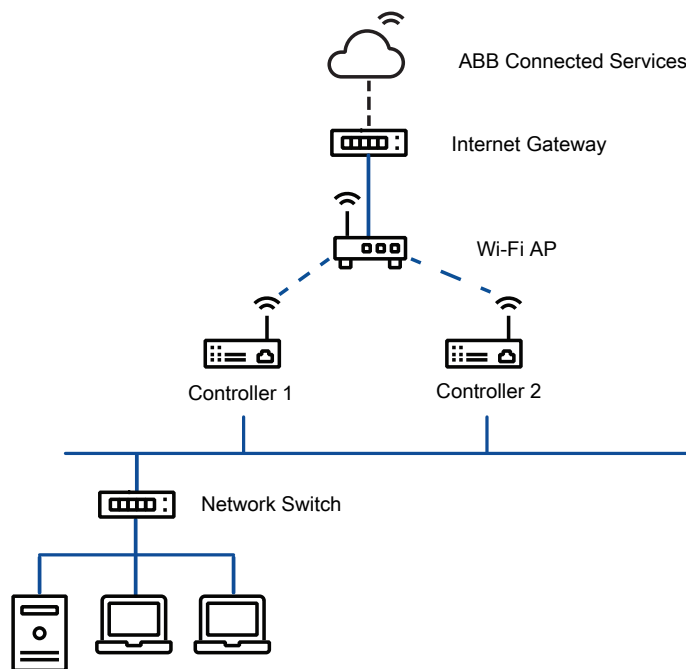
Continues on next page

Connection Type – Connected Services with Wi-Fi module

In the following scenario the Controller 1 and Controller 2 are connected with Connected Services Gateway Wi-Fi module. The Wi-Fi modules can be configured to connect with any of the available Wi-Fi access points. These access points must be enabled with internet access to reach ABB Cloud.

Refer to Connected Services Settings table in the following figure for detailed configuration. Based on this configuration Connected Services Gateway Wi-Fi module will connect to the internet enabled Wi-Fi network and reaches the ABB Cloud.

Connected Services is configured with ABB Connect Connection Type, which means all the communication to the ABB Cloud will pass and be routed through the Connected Services Gateway Wi-Fi module.



xx1900001183

Wi-Fi connection settings for controller 1 and 2	
State	Enabled
SSID	SSID-123
Key	1234567890
Security	Automatic

Connected services settings	
State	Enabled
Connection type	ABB Connect
Proxy used	Yes or No
Server polling	Fast or Slow
Connected services mode	Omnicores data collection

Continues on next page

2 RobotWare-OS

2.3.11 Network topology scenarios

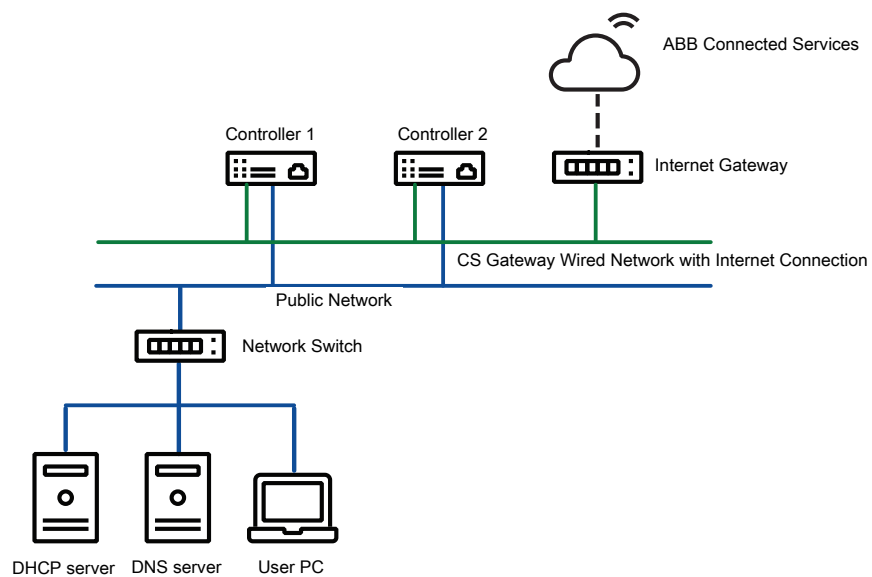
Continued

Connection Type – Connected Services with Wired module

In the following scenario the Controller 1 and Controller 2 are connected with Connected Services Gateway Wired module. Since it is a wired module it requires a wired connection with internet access to reach ABB Cloud. Controller 1 and 2 are also connected to public network which could be a factory network.

Wired module always should be configured with static IP. Refer to the Connected Services Gateway Wired settings table for a simple network configuration.

Connected Services is configured with the ABB Connect Connection Type, which means all the communication to the ABB Cloud will pass and be routed through the Connected Services Gateway Wired module and not on the public network.



xx1900001184

Wired connection settings	Controller 1 (NW1)	Controller 2
State	Enabled	Enabled
IP Address	192.168.200.20	192.168.200.21
Subnet mask	255.255.255.0	255.255.255.0
Default gateway	192.168.200.1	192.168.200.1
Preferred DNS	192.168.200.1	192.168.200.1
Secondary DNS	0.0.0.0	0.0.0.0

	Controller 1	Controller 2	Internet Gateway	DHCP server	DNS server	User PC
IP Address	172.16.16.100	172.16.16.101	192.168.200.1	172.16.16.3	172.16.16.2	172.16.16.110
Subnet mask	255.255.255.0	255.255.255.0	255.255.255.0	255.255.255.0	255.255.255.0	255.255.255.0
Default gateway	172.16.16.1	172.16.16.1		172.16.16.1	172.16.16.1	172.16.16.1

Continues on next page

Connected services settings	
State	Enabled
Connection type	ABB Connect
Proxy used	No
Server polling	Fast
Connected services mode	Omnicores data collection

2 RobotWare-OS

2.3.11 Network topology scenarios

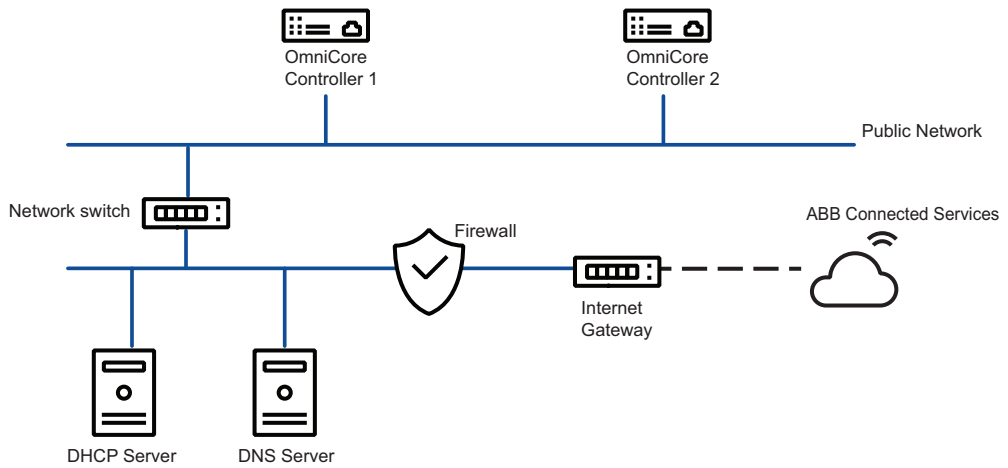
Continued

Connection Type – Public

If the factory network is enabled with internet access and firewalled, then the same network which is connected to the public port of the controller can be used to configure Connected Services.

In the following scenario the Controller 1 and Controller 2 are connected to public network by using public port of the controller which is the factory network enabled with internet. As a good practice the factory network must be firewalled for any unwanted inbound and outbound accesses if connected to internet.

Connected Services with Public Connection Type is configured with the Connected Services Settings in the following way:



xx230000120

	Controller 1	Controller 2	Internet Gate-way	DHCP server	DNS server
IP Address	172.16.16.100	172.16.16.102	172.16.16.1	172.16.16.3	172.16.16.2
Subnet mask	255.255.255.0	255.255.255.0	255.255.255.0	255.255.255.0	255.255.255.0
Default gate-way	172.16.16.1	172.16.16.1		172.16.16.1	172.16.16.1
DNS	172.16.16.2	172.16.16.2		172.16.16.2	

Connected services settings	
State	Enabled
Connection type	Public
Proxy used	No
Server polling	Fast
Connected services mode	Omnicores data collection



Note

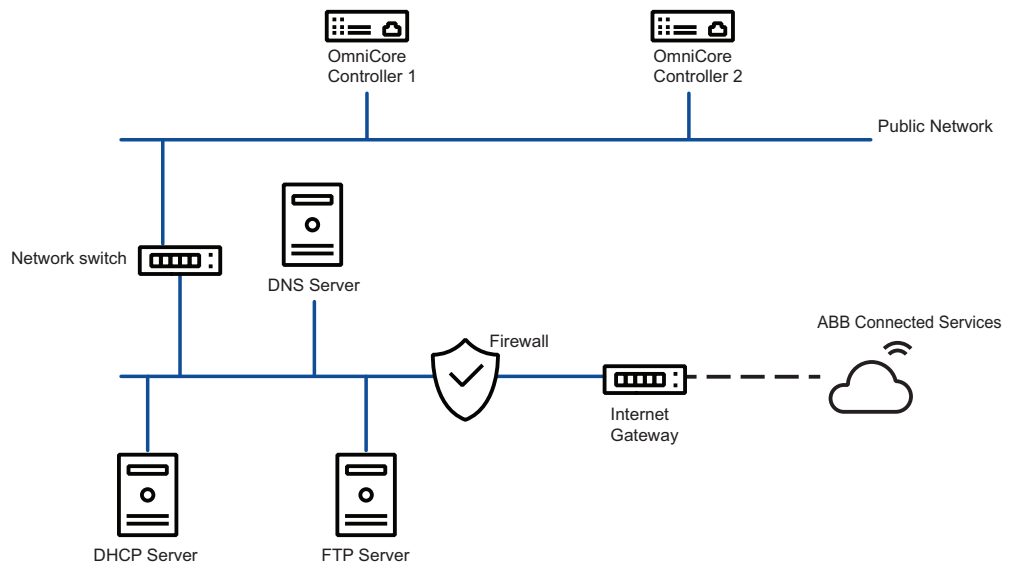
Ensure that there is a Firewall setup on customer network for internet access (outbound only). This needs to be confirmed also in the Omnicores Firewall settings.

Continues on next page

Connection Type – Public - With customer storage enabled

The following figure shows the scenario of using the customer storage to store controller data. Customer storage could be a controller local disk, mapped network disk (on ftp/sftp or nfs server) disk. In the following scenario we have used an ftp/sftp server as a customer storage in the factory network. This ftp/sftp server can be mounted as a local disk on the controller. So, during the Connected Services configuration the disk path should be mentioned as the mounted ftp/sftp disk. The disk path is ftp/sftp for mounted ftp/sftp disk and pc : for mounted network disk on the controller.

Connected Services with Public Connection Type is configured with the Connected Services Settings in the following way:



xx230000122

	Controller 1	Controller 2	Internet Gateway	DHCP server	DNS server	FTP server
IP Address	172.16.16.100	172.16.16.102	172.16.16.1	172.16.16.3	172.16.16.2	172.16.16.4
Subnet mask	255.255.255.0	255.255.255.0				
Default gateway	172.16.16.1	172.16.16.1				
DNS	172.16.16.2	172.16.16.2				

Connected services settings	
State	Enabled
Connection type	Public
Proxy used	No
Server polling	Fast
Connected services mode	Omnicores data collection
Customer storage	Disk
Disk path	ftp/

Continues on next page

2 RobotWare-OS

2.3.11 Network topology scenarios

Continued

Connected Services using ABB Gateway Service box

Overview

This section explains how Connected Services is configured using an external Internet gateway (3G/4G, Wi-Fi, and so on) not defined as default gateway in the controller. In this case connected services should be configured with the connection type custom.

The gateway service box can be connected on customer WAN port, management port, or Connected Services Gateway wired port.

Controller with DHCP

Use the following procedure to configure the Connected Services from the FlexPendant when there is controller with DHCP.

- 1 Open **Settings**.
- 2 Tap **ABB Connected Services**.
- 3 Tap **Connected Services** on the left pane.
The configuration parameters for connected services is displayed.
- 4 In the **Connection Type** list, tap and select **Custom**.
- 5 In the **Internet Gateway IP** field, type the IP address of internet gateway.
- 6 In the **Internet DNS IP** field, type the IP address of internet DNS.
- 7 Tap **Apply**.

The **Restart** confirmation message is displayed.

- 8 Click **OK**.

The controller is restarted.

The connected services start communicating to the server based on the configuration.

Check the connectivity status in the event logs. For more details, see [Connected Services information on page 77](#).

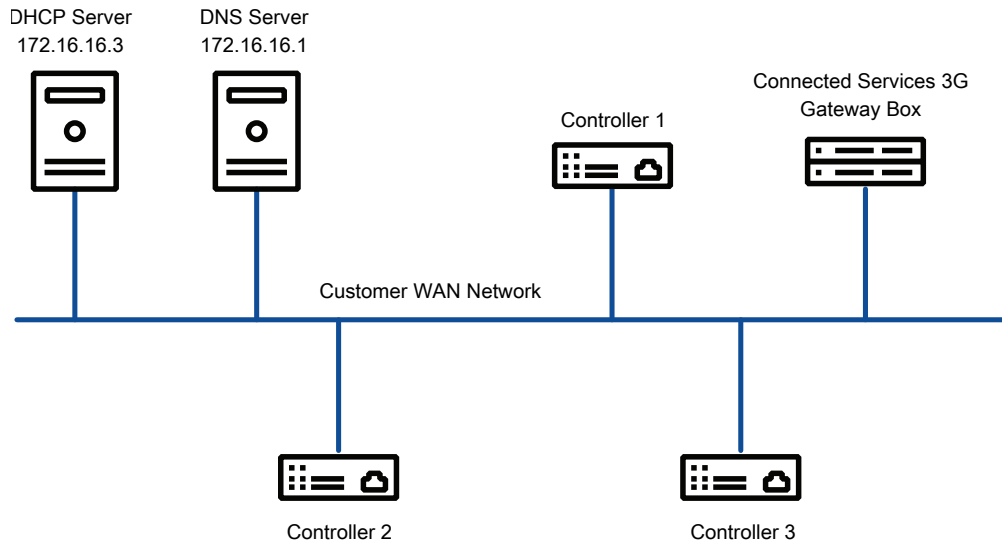
Gateway box on customer network

When gateway box is configured for multiple controllers, then the LAN IP of the gateway box must change according to the WAN IP network segment.

The gateway box should be connected to the customer network. And, the LAN IP should be modified to match with the customer network IP segment.

Continues on next page

The following figure and table show a typical network infrastructure:



xx1800002942

DHCP configuration	Controller 1	Controller 2	Controller 3
IP	172.16.16.58	172.16.16.59	172.16.16.60
Mask	255.255.255.0	255.255.255.0	255.255.255.0
CSE Enabled	Yes	Yes	Yes
Connection type	Custom	Custom	Custom
Internet Gateway IP	172.16.16.25	172.16.16.25	172.16.16.25
Internet DNS IP	172.16.16.25	172.16.16.25	172.16.16.25

For more information about how to do setting for the gateway box for multiple controllers, see *Product manual - Connected Services*.



Note

The network infrastructure is an example to demonstrate the network topology.



CAUTION

Ensure you always have Internet access with firewall.



Note

Using the ABB Service Box will allow Remote Access features. A standard 4G router can also be used with same principles.

Continues on next page

2 RobotWare-OS

2.3.11 Network topology scenarios

Continued

Connected Services using customer Gateway

Overview

This section explains how Connected Services is configured using an external Internet gateway (provided by the customer) not defined as default gateway in the controller. In this case connected services should be configured with the connection type custom.

The gateway service box can be connected on customer WAN port, management port, or Connected Services Gateway wired port.

Controller with DHCP

Use the following procedure to configure the Connected Services from the FlexPendant when there is controller with DHCP.

- 1 Open **Settings**.
- 2 Tap **ABB Connected Services**.
- 3 Tap **Connected Services** on the left pane.
The configuration parameters for connected services is displayed.
- 4 In the **Connection Type** list, tap and select **Custom**.
- 5 In the **Internet Gateway IP** field, type the IP address of internet gateway.
- 6 In the **Internet DNS IP** field, type the IP address of internet DNS.
- 7 Tap **Apply**.

The **Restart** confirmation message is displayed.

- 8 Click **OK**.

The controller is restarted.

The connected services start communicating to the server based on the configuration.

Check the connectivity status in the event logs. For more details, see [Connected Services information on page 77](#).

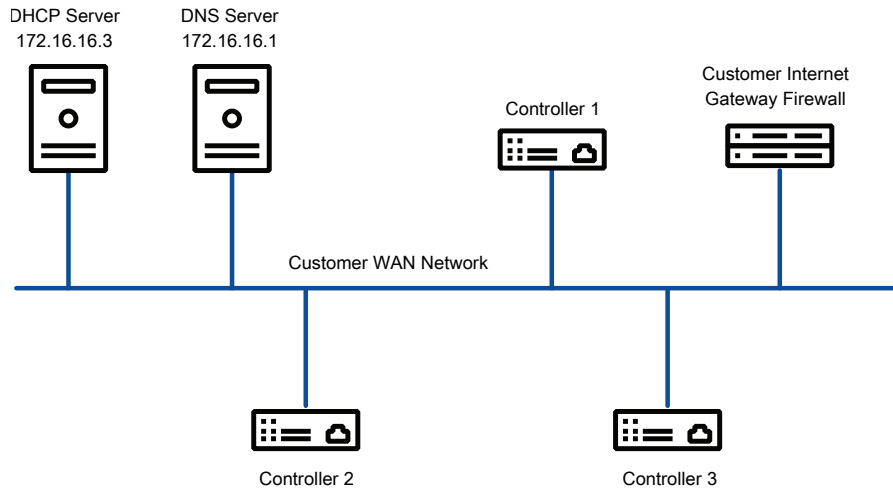
Gateway box on customer network

When gateway box is configured for multiple controllers, then the LAN IP of the gateway box must change according to the WAN IP network segment.

The gateway box should be connected to the customer network. And, the LAN IP should be modified to match with the customer network IP segment.

Continues on next page

The following figure and table show a typical network infrastructure:



xx2300001231

DHCP configuration	Controller 1	Controller 2	Controller 3
IP	172.16.16.58	172.16.16.59	172.16.16.60
Mask	255.255.255.0	255.255.255.0	255.255.255.0
CSE Enabled	Yes	Yes	Yes
Connection type	Custom	Custom	Custom
Internet Gateway IP	172.16.16.25	172.16.16.25	172.16.16.25
Internet DNS IP	172.16.16.25	172.16.16.25	172.16.16.25



Note

The network infrastructure is an example to demonstrate the network topology.



CAUTION

Ensure you always have Internet access with firewall.



Note

It is possible to use an ABB Service Box which will also allow Remote Access features. A standard 4G router can also be used with same principles.



Note

When you use an internet gateway make sure it is firewalled to allow only minimum required outbound commands.

2 RobotWare-OS

2.4.1 Cyclically evaluated logical conditions

2.4 Cyclic bool

2.4.1 Cyclically evaluated logical conditions

Purpose

The purpose of cyclically evaluated logical conditions, *Cyclic bool*, is to allow a RAPID programmer to connect a logical condition to a persistent boolean variable. The logical condition will be evaluated every 12 ms and the result will be written to the connected variable.

What is included

The RobotWare base functionality *Cyclic bool* includes:

- **instructions for setting up *Cyclic bool*:** SetupCyclicBool, RemoveCyclicBool, RemoveAllCyclicBool
- **functions for retrieving the status of *Cyclic bool*:** GetMaxNumberOfCyclicBool, GetNextCyclicBool, GetNumberOfCyclicBool.

Basic approach

This is the general approach for using *Cyclic bool*. For more detailed examples of how this is done, see [Cyclic bool examples on page 107](#).

- 1 **Declare a persistent boolean variable, for example:**

```
PERS bool cyclicbool1;
```

- 2 **Connect a logical condition to the variable, for example:**

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

- 3 **Use the variable when programming, for example:**

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

- 4 **Remove connection when no longer useful, for example:**

```
RemoveCyclicBool cyclicbool1;
```

Restart and reset behavior

The table below describes the functionality of *Cyclic bool* when the program pointer is moved or when the controller is restarted.

Action	Description
Program pointer to main	The behavior when the program pointer is set to main is configurable, see Configuration on page 105 .
Restart or power fail	This will have no effect. All connected <i>Cyclic bool</i> conditions will remain and the evaluation will be restarted immediately.
Reset RAPID	This will remove all connected <i>Cyclic bool</i> conditions.
Reset system	

Continues on next page

Configuration

The following behavior of the Cyclic bool functionality can be configured:

Parameter	Description
<i>RemoveAtPpToMain</i>	It is possible to configure if the cyclically evaluated logical conditions shall be removed or not when setting the program pointer to <code>main</code> . <ul style="list-style-type: none"> • <i>On</i> - remove. • <i>Off</i> - do not remove (default behavior).
<i>ErrorMode</i>	It is possible to configure which error mode to use when the evaluation of a Cyclic bool fails. <ul style="list-style-type: none"> • <i>SysStopError</i>ⁱ - stop RAPID execution and produce an error log (default behavior). • <i>Warning</i> - produce a warning log. • <i>None</i> - do nothing.
<i>RecoveryMode</i>	It is possible to configure if a failing Cyclic bool shall be recovered or not. <ul style="list-style-type: none"> • <i>On</i> - try to recover the evaluation of a failing Cyclic bool (default behavior). • <i>Off</i> - do not try to recover the evaluation of a Cyclic bool.

ⁱ Error mode *SysStopError* can only be combined with *RecoveryMode* - "On".

For more information, see [System parameters on page 110](#).

Syntax

`SetupCyclicBool` Flag Cond [`\Signal`]

Flag shall be of:

- Data type: `bool`
 - Object type: `PERS` or `TASK PERS`

Cond shall be a `bool` expression that may consist of:

- Data types: `num`, `dnum` and `bool`
 - Object type: `PERS`, `TASK PERS`, or `CONST`
- Data types: `signal``di`, `signal``do` or physical `di` and `do`
 - Object type: `VAR`
- Operands: `'NOT'`, `'AND'`, `'OR'`, `'XOR'`, `'='`, `'('`, `'('`

`\Signal` shall be of:

- Object type: `signal``do`

`RemoveCyclicBool` Flag

Flag shall be of:

- Data type: `bool`
 - Object type: `PERS` or `TASK PERS`

Limitations

- Records and arrays are not allowed in the logical condition.
- A maximum of 60 conditions can be connected at the same time.

Continues on next page

2 RobotWare-OS

2.4.1 Cyclically evaluated logical conditions

Continued

- **Any PERS num or dnum, CONST num or dnum or literal num or dnum used in a condition must be of integer type. If using any decimal value this will cause a fatal error.**

2.4.2 Cyclic bool examples

Using digital input and output signals

```

! 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

```

Using bool variables

```

! 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

```

Using num and dnum variables

```

! 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

```

Continues on next page

2 RobotWare-OS

2.4.2 Cyclic bool examples

Continued

Using alias variables

```
! 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
```

Using user defined constants for comparison

```
! 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
```

Continues on next page

Handing over arguments by reference

If the instruction `SetupCyclicBool` is used inside a called procedure, it is possible to hand over conditions as arguments to that procedure.

Using conditions passed by reference works only for `SetupCyclicBool`. Conditions passed by reference has the same restrictions as conditions for `SetupCyclicBool`.

This functionality works regardless if the modules are `Nostepin` or has any other module attributes.

```
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 RobotWare-OS

2.4.3 System parameters

2.4.3 System parameters

About the system parameters

This is a brief description of the system parameters used by *Cyclic bool*. For more information about the parameters, see *Technical reference manual - System parameters*.

Type *Cyclic bool* settings

The system parameters used by *Cyclic bool* belong to the type *Cyclic bool settings* in topic *Controller*.

Parameter	Description
<i>Name</i>	There can be only one instance of each allowed value, that is a maximum of three instances in the system. All three instances will be installed in the system (default) and cannot be removed. <ul style="list-style-type: none">• <i>RemoveAtPpToMain</i>• <i>ErrorMode</i>• <i>RecoveryMode</i>
<i>RemoveAtPpToMain</i>	The action value <i>RemoveAtPpToMain</i> is used to configure if a connected <i>Cyclic bool</i> shall be removed or not when setting the program pointer to Main.
<i>ErrorMode</i>	The action value <i>ErrorMode</i> is used to configure which error mode to use when evaluation fails.
<i>RecoveryMode</i>	The action value <i>RecoveryMode</i> is used to configure which recovery mode to use when evaluation fails.

2.4.4 RAPID components

About the RAPID components

This is an overview of all RAPID instructions, functions, and data types in *Cyclic bool*.

For more information, see *Technical reference manual - RAPID Instructions, Functions and Data types*

Instructions

Instruction	Description
SetupCyclicBool	SetupCyclicBool connects a logical condition to a boolean variable.
RemoveCyclicBool	RemoveCyclicBool removes a specific connected logical condition.
RemoveAllCyclicBool	RemoveAllCyclicBool removes all connected logical conditions.

Functions

Function	Description
GetMaxNumberOfCyclicBool	GetMaxNumberOfCyclicBool retrieves the maximum number of cyclically evaluated logical condition that can be connected at the same time.
GetNextCyclicBool	GetNextCyclicBool retrieves the name of a connected cyclically evaluated logical condition.
GetNumberOfCyclicBool	GetNumberOfCyclicBool retrieves the number of a connected cyclically evaluated logical condition.
IsCyclicBool	IsCyclicBool is used to test if a persistent boolean is a Cyclic bool or not, i.e. if a logical condition has been connected to the persistent boolean variable with the instruction SetupCyclicBool.

Data types

Cyclic bool includes no data types.

2 RobotWare-OS

2.5.1 Introduction to Device Command Interface

2.5 Device Command Interface

2.5.1 Introduction to Device Command Interface

Purpose

Device Command Interface provides an interface to communicate with I/O devices on industrial networks.

This interface is used together with raw data communication, see [Raw data communication on page 139](#).

What is included

The RobotWare base functionality Device Command Interface gives you access to:

- Instruction used to create a DeviceNet header.
-

Basic approach

This is the general approach for using Device Command Interface. For a more detailed example of how this is done, see [Write rawbytes to DeviceNet on page 114](#).

- 1 Add a DeviceNet header to a `rawbytes` variable.
 - 2 Add the data to the `rawbytes` variable.
 - 3 Write the `rawbytes` variable to the DeviceNet I/O.
 - 4 Read data from the DeviceNet I/O to a `rawbytes` variable.
 - 5 Extract the data from the `rawbytes` variable.
-

Limitations

Device command communication require the option for the industrial network in question.

Device Command Interface is supported by the following type of industrial networks:

- DeviceNet
- EtherNet/IP

2.5.2 RAPID components and system parameters

Data types

There are no RAPID data types for Device Command Interface.

Instructions

This is a brief description of each instruction in Device Command Interface. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
PackDNHeader	PackDNHeader adds a DeviceNet header to a rawbytes variable. The header specifies a service to be done (e.g. set or get) and a parameter on a DeviceNet I/O device.

Functions

There are no RAPID functions for Device Command Interface.

System parameters

There are no specific system parameters in Device Command Interface. For information on system parameters in general, see *Technical reference manual - System parameters*.

2.5.3 Code example

Write rawbytes to DeviceNet

In this example, data packed as a `rawbytes` variable is written to a DeviceNet I/O device. For more details regarding `rawbytes`, see [Raw data communication on page 139](#).

```
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;
```

Continues on next page

```
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 RobotWare-OS

2.6.1 Overview

2.6 Electronically Linked Motors

2.6.1 Overview

Description

Electronically Linked Motors makes a master/follower configuration of motors (for example two additional axes). The follower axis will continuously follow the master axis in terms of position, velocity, and acceleration.

For stiff mechanical connection between the master and followers, the torque follower function can be used. Instead of regulating to exactly the same position for the master and follower, the follower axis will get its torque reference as a quota of the torque of the master axis. A small position error between master and follower will occur depending on backlash and mechanical misalignment.

Purpose

The primary purpose of Electronically Linked Motors is to replace driving shafts of gantry machines, but the base functionality can be used to control any other set of motors as well.

What is included

The RobotWare base functionality Electronically Linked Motors gives you access to:

- a service routine for defining linked motor groups and trimming the axis positions
 - system parameters used to configure a follower axis
-

Basic approach

This is the general approach for setting up Electronically Linked Motors. For a more detailed description of how this is done, see the respective section.

- 1 Configure the additional axes as a mechanical unit. See *Application manual - Additional axes*.
 - 2 Configure tolerance limits in the system parameters, in the types *Linked M Process*, *Process*, and *Joint*.
 - 3 Restart the controller for the changes to take effect.
 - 4 Set values to data variables, defining the linked motor group and connecting follower and master axes.
 - 5 Use the service routine to trim positions or reset follower after position error.
-

Limitations

There can be up to 5 follower axes. The follower axes can be configured to follow one master each, or several followers can follow one master, but the total number of follower axes cannot be more than 5.

The follower axis cannot be an ABB robot (IRB robot). The master axis can be either an additional axis or a robot axis.

Continues on next page

The RAPID instruction `IndReset` (*Independent Reset*) cannot be used in combination with Electronically Linked Motors.

2 RobotWare-OS

2.6.2.1 System parameters

2.6.2 Configuration

2.6.2.1 System parameters

About the system parameters

This is a brief description of each parameter used for the option *Electronically Linked Motors*. For more information, see the respective parameter in *Technical reference manual - System parameters*.

Joint

These parameters belong to the topic *Motion* and the type *Joint*.

Parameter	Description
Follower to Joint	Specifies which master axis this axis shall follow. Refers to the parameter <i>Name</i> in the type <i>Joint</i> . Robot axes are referred to as rob1 followed by underscore and the axis number (for example rob1_6).
Use Process	Id name of the process that is called. Refers to the parameter <i>Name</i> in the type <i>Process</i> .
Lock Joint in Ipol	A flag that locks the axis so it is not used in the path interpolation. This parameter must be set to TRUE when the axis is electronically linked to another axis.

Process

These parameters belong to the topic *Motion* and the type *Process*.

Parameter	Description
Name	Id name of the process.
Use Linked Motor Process	Id name of electronically linked motor process. Refers to the parameter <i>Name</i> in the type <i>Linked M Process</i> .

Linked M Process

These parameters belong to the topic *Motion* and the type *Linked M Process*.

Parameter	Description
Name	Id name for the linked motor process.
Offset Adjust Delay Time	Time delay from control on until the follower starts to follow the master. This can be used to give the master time to stabilize before the follower starts following.
Max Follower Offset	The maximum allowed difference in distance (in radians or meters) between master and follower. If <i>Max Follower Offset</i> is exceeded, emergency stop is activated.
Max Offset Speed	The maximum allowed difference in speed (in rad/s or m/s) between master and follower. If <i>Max Offset Speed</i> is exceeded, emergency stop is activated.
Offset Speed Ratio	Defines how large part of the <i>Max Offset Speed</i> that can be used to compensate for position error.

Continues on next page

Parameter	Description
Ramp Time	Time for acceleration up to <i>Max Offset Speed</i> . The proportion constant for position regulation is ramped from zero up to its final value (<i>Master Follower kp</i>) during <i>Ramp Time</i> .
Master Follower kp	The proportion constant for position regulation. Determines how fast the position error is compensated.
Torque follower	Set to True if the follower and master should share torque instead of regulating on exact position.
Torque quota	The follower axis will get its torque reference according to the master axis torque multiplied with the quota. If motors and gears are identical, it is recommend to set this value a bit lower than 1, for example, 0.95.

2 RobotWare-OS

2.6.2.2 Configuration example

2.6.2.2 Configuration example

About this example

This is an example of how to configure the additional axis M8DM1 to be a follower to the axis M7DM1 and axis M9DM1 to be a follower to robot axis 6.

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

2.6.3 Managing a follower axis

2.6.3.1 Using the service routine for a follower axis

About the service routine

When the follower axis is configured as a mechanical unit and connected to a master axis, the service routine can be used to:

- calibrate the follower axis
- reset follower after a position error
- tune a torque follower axis, see [Tuning a torque follower on page 126](#).

Copy service routine file to HOME

Copy the file *linked_m.sys* from the folder *.../RobotControl_7.xxx/utility/*, to the *HOME* folder of the robot system.

Load cfg files

Load the configuration files *LINKED_M_MMC.cfg* and *LINKED_M_SYS.cfg*. These are located in the directory:

...utility\LinkedMotors.

Loading configuration files can be done with RobotStudio. How to do this is described in:

Tool	Description of loading cfg files
RobotStudio	Section <i>Loading a configuration file</i> in <i>Operating manual - RobotStudio</i> .

Restart the controller after loading the configuration files.

Data variables

When the service routine starts, it will read values from system parameters and set the values for a set of data variables used by the service routine. These variables only need to be set manually if something goes wrong, see [Data setup on page 130](#).

Start service routine



Note

The controller must be in manual or auto mode to run this service routine.

Step	Action
1	On the start screen, tap Operate , and then select Service Routines from the menu.
2	Select Linked_m and tap Go to . If Linked_m is not listed, browse to the folder where you placed it.
3	Press and hold the enabling device.
4	Press the RUN button to start the service routine.
5	Tap Menu 1 . The follower axes that are set up in the system are shown in the task bar.

Continues on next page

2 RobotWare-OS

2.6.3.1 Using the service routine for a follower axis

Continued

Step	Action
6	Tap the follower axis you want to use the service routine for. The main menu of the service program is now shown.

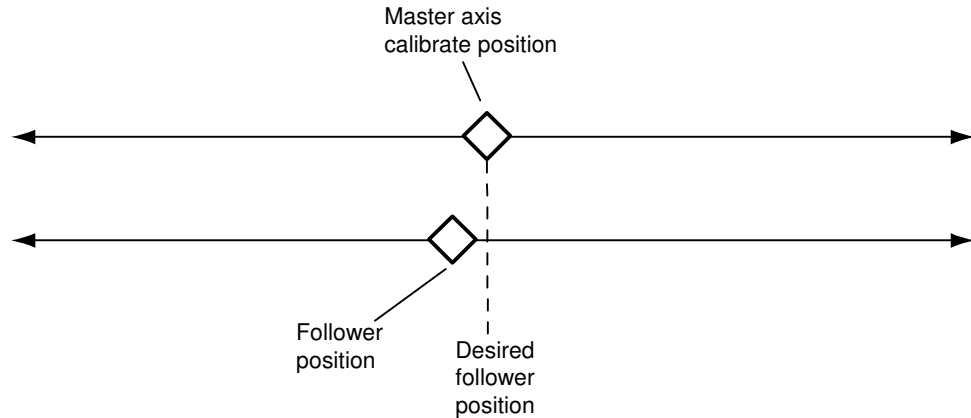
Menu buttons

Button	Description
AUTO	Automatically moves the follower axis to the position corresponding to the master axis, see Reset follower automatically on page 125 .
STOP	Stops the movement of the follower axis. Can be used when jogging or using AUTO and the movement must be stopped immediately.
JOG	Manual stepwise movement of the follower axis, see Jog follower axis on page 123 . If the follower axis is synchronized with the master axis, it will resume its position when you tap AUTO or when you exit the service program.
UNSYNC	Used to suspend the synchronization between follower axis and master axis, see Unsyncronize on page 123 .
HELP	Show some help for how to use the service program. The button Next shows the next help subject.

2.6.3.2 Calibrate follower axis position

Overview

Before the follower axis can follow the master axis, you must define the calibration positions for both master and follower.



en0400000963

This calibration is done by following the procedures below:

- 1 Jog the master axis to its calibration position.
- 2 Unsynchronize the follower and master axes. See [Unsynchronize on page 123](#).
- 3 Jog the follower to the desired position. See [Jog follower axis on page 123](#).
- 4 Fine calibrate follower axis. See [Fine calibrate on page 124](#).

Unsynchronize

Step	Action
1	In the main menu of the service routine, tap UNSYNC .
2	Confirm that you want to unsynchronize the axes by tapping YES .
3	Restart the controller when an information text tells you to do it. After the restart the follower axis is no longer synchronized with the master axis.

Jog follower axis

Step	Action
1	In the main menu of the service program, tap JOG .
2	Select the speed with which the follower axis should move when you jog it.
3	Select the step size with which the follower axis should move for each step you jog it.
4	Tap on Positive or Negative , depending on in which direction you want to move the follower axis. Jog the follower axis until it is exactly in the calibration position (the position that corresponds to the master axis calibration position).

Continues on next page

2 RobotWare-OS

2.6.3.2 Calibrate follower axis position

Continued

Fine calibrate

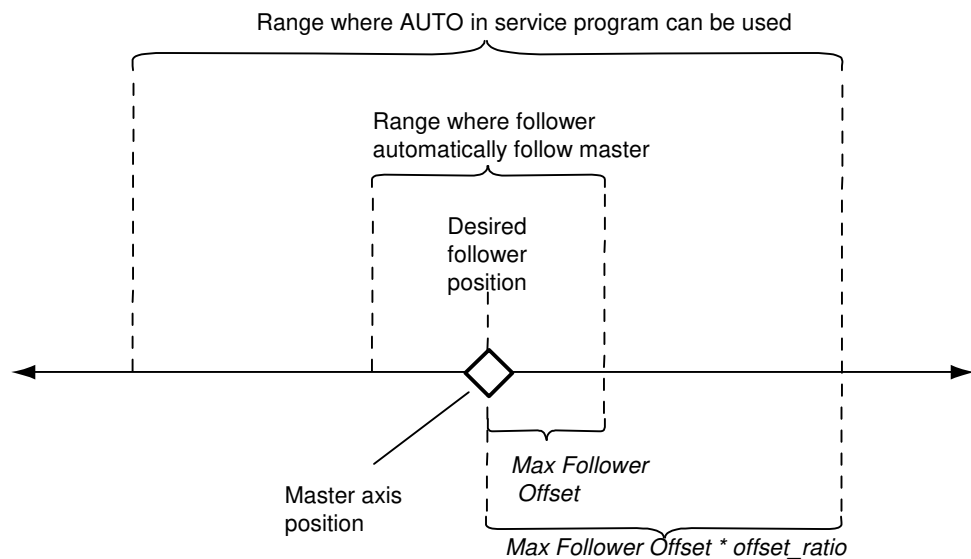
Step	Action
1	Open the Calibrate app.
2	Select the mechanical unit that the follower axis belongs to.
3	Tap the button Calib. Parameters .
4	Tap Fine Calibration....
5	In the warning dialog that appears, tap Yes .
6	Select the axis that is used as follower axis and tap Calibrate .
7	In the warning dialog that appears, tap Calibrate . The follower axis is now calibrated. As soon as the follower is calibrated, it is also synchronized with the master again.

2.6.3.3 Reset follower axis

Overview

If the follower offset exceeds its tolerance limits (configured with the system parameter *Max follower offset*), the service routine must be used to move the follower back within the tolerance limits. This can be done automatically in the service routine if the follower is within the AUTO range. Otherwise the follower must be manually jogged.

The range where AUTO can be used is determined by the system parameter *Max Follower Offset* multiplied with the data variable *offset_ratio*.



en0400000962

Reset follower automatically

Step	Action
1	In the main menu of the service routine, tap AUTO .
2	Select the speed with which the follower axis should move to its desired position.

Reset follower by manual jogging

Step	Action
1	In the main menu of the service routine, tap JOG .
2	Select the speed with which the follower axis should move when you jog it.
3	Select the step size with which the follower axis should move for each step you jog it.
4	Tap on Positive or Negative , depending on which direction you want to move the follower axis. Jog the follower until it is within the tolerance of <i>Max Follower Offset</i> (or use AUTO when you are close enough).

2 RobotWare-OS

2.6.4.1 Description of torque follower

2.6.4 Tuning a torque follower

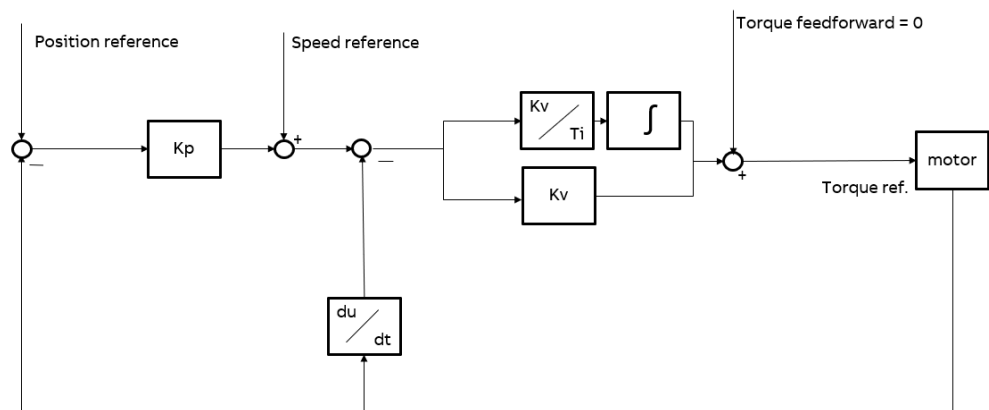
2.6.4.1 Description of torque follower

About torque follower

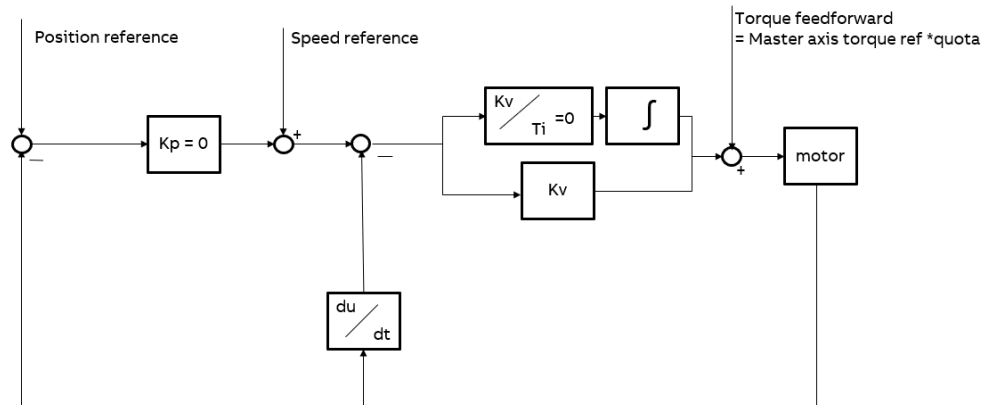
The follower axis can be set up so that it gets its torque reference directly from the master axis.

The following simplified graphics shows the control loop for both master and follower axis.

Master axis



Follower axis



Description

The follower axis gets its torque feedforward according to a quota of torque ref on the master axis.

When the follower axis is set as torque follower, the control parameters position control gain (K) and speed loop integral part (Kv/Ti) is internally set to zero. It is recommended to run the follower axis with a small value of Kv and *FFW mode = Spd*.

Continues on next page

Torque quota

The sharing of torque is done so that the follower axis receives its torque reference as the master reference multiplied with the torque quota. The setup is done with equal motors and gears quota, and the ideal would be 1.0, but for stability reasons it is recommended to set it to a value below that, for example, 0.95.

If motor and gearbox differs between master and follower, manual calculation is required to achieve good sharing and stability.

The torque quota is configured with the system parameter *Torque quota*, see *Technical reference manual - System parameters*, topic *Motion*, type *Linked M Process*.

Example of torque follower configuration

An example of a torque follower configuration (part of MOC.cfg):

```
LINKED_M_PROCESS:
  -name "TorqueFollower" -offset_adj_delay_time 0.1 -max_offset
    50\
  -max_offset_speed 1 -offset_speed_ratio 0.3 -ramp_time 0.1\
  -kp_offset 0.2 -torque_follower -torque_distribution 0.9

LCM0:
  -name "MasterAxis" -Kp 10 -Kv 0.15 -Ti 0.2 -ffw_mode 1
  -name " TorqueFollower " -Kp 10 -Kv 0.05 -Ti 0.2 -ffw_mode 1
```

When enabling `torque_follower`, the parameters marked with bold font in the example, are no longer used. `max_offset` is still valid but should be increased. Normally the follower axis should use a smaller value of `Kv`, but it is recommend to still have some natural damping.

**Note**

It is highly recommended to use same type of motor and gearbox for the master and follower.

When the torque follower is used with different motor types or gearboxes, the quota must be set accordingly. The quota is shared on motor torque and does not consider gear ratio.

2 RobotWare-OS

2.6.4.2 Using the service routine to tune a torque follower

2.6.4.2 Using the service routine to tune a torque follower

About the service routine for torque follower

The service routine *Linked_M* can be used to find suitable values of some parameters for torque follower configuration. When the values are found, the system parameters are updated and a new fine calibration is done. After that, there is no need for any tuning of the torque follower.

Opening the tune torque follower menu

	Action	Illustration
1	Start the service routine (as described by the first steps in Start service routine on page 121).	
2	Tap Menu 2 .	
3	Tap on the name of the follower axis to tune.	
4	Use the tune torque follower menu as described below.	

Tuning the torque quota

Use this procedure to change the torque quote for the follower axis.

	Action	Illustration
1	Tap Torque quota .	
2	Type a number (between 0 and 10) for the follower's quota master torque. For example, 0.95 will result in 95% of the master torque will be set to the follower. Values above 1 should only be used in case the follower axis is dimensioned with higher torque possibilities.	
3	To update the system parameters using the new value, tap Store to cfg . If not saved to cfg, the new value will be used until the robot controller is restarted, but the value will be lost at restart.	

Tuning the temporary position delta

Use this procedure to tune the position delta of the torque follower axis. This delta value is then used to adjust the fine calibration of the follower axis.

	Action	Illustration
1	Tap Temp. position delta .	
2	Type a number (degrees on motor side) that will be added to the position reference for the follower axis.	

Continues on next page

2.6.4.2 Using the service routine to tune a torque follower

Continued

	Action	Illustration
3	Test which value results in the lowest torque tension and make a fine calibration of the master axis. This will update the follower axis with the current position delta.	

2 RobotWare-OS

2.6.5.1 Set up data for the service routine

2.6.5 Data setup

2.6.5.1 Set up data for the service routine

Overview

At start of the service routine for Electronically Linked Motors, some data variables are read from the linked motor configuration. These variables are used by the service routine. If they are not read correctly, the variables need to be edited in the service routine.

Data descriptions

Data variable	Description
<code>l_f_axis_name</code>	A name for the follower axis that will be displayed on the FlexPendant. String array with 5 elements, one for each follower axis. If you only have one linked motor, use only the first element.
<code>l_f_mecunt_n</code>	The name of the mechanical unit for the follower axis. Refers to the system parameter <i>Name</i> in the type <i>Mechanical Unit</i> . String array with 5 elements, one for each follower axis. If you only have one linked motor, use only the first element.
<code>l_f_axis_no</code>	Defines which axis in the mechanical unit (<code>l_f_mecunt_n</code>) is the follower axis. Num array with 5 elements, one for each follower axis. If you only have one linked motor, use only the first element.
<code>l_m_mecunt_n</code>	The name of the mechanical unit for the master axis. Refers to the system parameter <i>Name</i> in the type <i>Mechanical Unit</i> . String array with 5 elements, one for each master axis. If you only have one linked motor, use only the first element.
<code>l_m_axis_no</code>	Defines which axis in the mechanical unit (<code>l_m_mecunt_n</code>) is the master axis. Num array with 5 elements, one for each master axis. If you only have one linked motor, use only the first element.
<code>offset_ratio</code>	Defines the range where the AUTO function in the service program reset the follower axis. <code>offset_ratio</code> defines this range as a multiple of the range where the follower automatically follow the master (defined with the parameter <i>Max Follow Offset</i>). If the follower has a position error that is larger than <i>Max Follower Offset</i> * <code>offset_ratio</code> , the follower must be reset manually. For more information, see Reset follower axis on page 125 .
<code>speed_ratio</code>	Defines the speed of the follower axis when controlled by the service program. The values are given as a part of the maximum allowed manual speed (that is, the value 0.5 means half the max manual speed). Num array with 20 elements. Elements 1-5 define the speed "very slow" for each follower axis. Elements 6-10 define "slow", elements 11-15 define "normal" and elements 16-20 define "fast". If you only have one linked motor, use only elements 1, 6, 11 and 16.

Continues on next page

Data variable	Description
displacement	Defines the distance the follower axis will move for each tap on Positive or Negative when jogging the follower axis from the service program. The values are given in degrees or meters, depending on if the follower axis is circular or linear. Num array with 20 elements. Elements 1-5 define the displacement "very short" for each follower axis. Elements 6-10 define "short", elements 11-15 define "normal" and elements 16-20 define "long". If you only have one linked motor, use only elements 1, 6, 11 and 16.

Edit data variables

If the data needs to be edited, use the **Program Data** app on the FlexPendant. See *Operating manual - OmniCore* for more information on how to edit data.

2 RobotWare-OS

2.6.5.2 Example of data setup

2.6.5.2 Example of data setup

About this example

This is an example of how to set up the data variables for two follower axis. The first follower axis is M8C1B1, which is a follower to the additional axis M7C1B1. The second follower axis is M9C1B1, which is a follower to robot axis 6.

I_f_axis_name

Represented axis	Element and value in I_f_axis_name
Follower 1	{1}: "follow_external"
Follower 2	{2}: "follow_axis6"
Follower 3	{3}: ""
Follower 4	{4}: ""
Follower 5	{5}: ""

I_f_mecunt_n

Represented axis	Element and value in I_f_mecunt_n
Follower 1	{1}: "M8DM1"
Follower 2	{2}: "M9DM1"
Follower 3	{3}: ""
Follower 4	{4}: ""
Follower 5	{5}: ""

I_f_axis_no

Represented axis	Element and value in I_f_axis_no
Follower 1	{1}: 1
Follower 2	{2}: 1
Follower 3	{3}: 0
Follower 4	{4}: 0
Follower 5	{5}: 0

I_m_mecunt_n

Represented axis	Element and value in I_m_mecunt_n
Master 1	{1}: "M7DM1"
Master 2	{2}: "rob1"
Master 3	{3}: ""
Master 4	{4}: ""
Master 5	{5}: ""

Continues on next page

l_m_axis_no

Represented axis	Element and value in l_m_axis_no
Master 1	{1}: 1
Master 2	{2}: 6
Master 3	{3}: 0
Master 4	{4}: 0
Master 5	{5}: 0

offset_ratio

Represented axis	Element and value in offset_ratio
Follower 1	{1}: 10
Follower 2	{2}: 15
Follower 3	{3}: 0
Follower 4	{4}: 0
Follower 5	{5}: 0

speed_ratio

Represented axis	very slow	slow	normal	fast
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

Represented axis	very short	short	normal	long
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 RobotWare-OS

2.7.1 Introduction to file and I/O device handling

2.7 File and I/O device handling

2.7.1 Introduction to file and I/O device handling

About file and I/O device handling

The RobotWare file and I/O device handling gives the robot programmer control of files and fieldbuses from the RAPID code. This can, for example, be useful for:

- Reading from a bar code reader.
- Writing production statistics to a log file or to a printer.
- Transferring data between the robot and a PC.

The functionality for file and I/O device handling can be divided into groups:

Functionality group	Description
Binary and character based communication	Basic communication functionality. Communication with binary or character based files or I/O devices.
Raw data communication	Data packed in a container. Especially intended for fieldbus communication.
File and directory management	Browsing and editing of file structures.

2.7.2 Binary and character based communication

2.7.2.1 Overview

Purpose

The purpose of binary and character based communication is to:

- store information in a remote memory or on a remote disk
- let the robot communicate with other devices

What is included

To handle binary and character based communication, RobotWare gives you access to:

- instructions for manipulations of a file or I/O device
- instructions for writing to file or I/O device
- instruction for reading from file or I/O device
- functions for reading from file or I/O device.

Basic approach

This is the general approach for using binary and character based communication. For a more detailed example of how this is done, see [Code examples on page 137](#).

- 1 Open a file or I/O device.
- 2 Read or write to the file or I/O device.
- 3 Close the file or I/O device.

Limitations

Access to files and I/O devices cannot be performed from different RAPID tasks simultaneously. Such an access is performed by all instruction in binary and character based communication, as well as `WriteRawBytes` and `ReadRawBytes`. E.g. if a `ReadBin` instruction is executed in one task, it must be ready before a `WriteRawBytes` can execute in another task.

2 RobotWare-OS

2.7.2.2 RAPID components

2.7.2.2 RAPID components

Data types

This is a brief description of each data type used for binary and character based communication. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
iodev	iodev contains a reference to a file or I/O device. It can be linked to the physical unit with the instruction <code>Open</code> and then used for reading and writing.

Instructions

This is a brief description of each instruction used for binary and character based communication. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
Open	Open is used to open a file or I/O device for reading or writing.
Close	Close is used to close a file or I/O device.
Rewind	Rewind sets the file position to the beginning of the file.
Write	Write is used to write to a character based file or I/O device.
WriteBin	WriteBin is used to write a number of bytes to a binary I/O device or file.
WriteStrBin	WriteStrBin is used to write a string to a binary I/O device or file.
WriteAnyBin	WriteAnyBin is used to write any type of data to a binary I/O device or file.
ReadAnyBin	ReadAnyBin is used to read any type of data from a binary I/O device or file.

Functions

This is a brief description of each function used for binary and character based communication. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
ReadNum	ReadNum is used to read a number from a character based file or I/O device.
ReadStr	ReadStr is used to read a string from a character based file or I/O device.
ReadBin	ReadBin is used to read a byte (8 bits) from a file or I/O device. This function works on both binary and character based files or I/O devices.
ReadStrBin	ReadStrBin is used to read a string from a binary I/O device or file.

2.7.2.3 Code examples

Communication with character based file

This example shows writing and reading to and from a character based file. The line "The number is :8" is written to FILE1.DOC. The contents of FILE1.DOC is then read and the output to the FlexPendant is "The number is :8" followed by "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
```

Communication with binary file

In this example, the string "Hello", the current robot position and the string "Hi" is written to the binary file.

```
PROC write_bin_chan()
  VAR iodev file1;
  VAR num out_buffer{20};
  VAR num input;
  VAR robtarget target;

  Open "HOME:" \File:= "FILE1.DOC", file1 \Bin;

  ! Write control character eng
  out_buffer{1} := 5;
  WriteBin file1, out_buffer, 1;

  ! Wait for control character ack
  input := ReadBin (file1 \Time:= 0.1);
  IF input = 6 THEN
    ! Write "Hello" followed by new line
    WriteStrBin file1, "Hello\0A";
```

Continues on next page

2 RobotWare-OS

2.7.2.3 Code examples

Continued

```
! Write current robot position
target := CRobT(\Tool:= tool1\WObj:= wobj1);
WriteAnyBin file1, 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 file
WriteBin file1, out_buffer, 5;
ENDIF
Close file1;
ENDPROC
```

2.7.3 Raw data communication

2.7.3.1 Overview

Purpose

The purpose of raw data communication is to pack different type of data into a container and send it to a file or I/O device, and to read and unpack data. This is particularly useful when communicating via a fieldbus, such as DeviceNet.

What is included

To handle raw data communication, RobotWare gives you access to:

- instructions used for handling the contents of a `rawbytes` variable
 - instructions for reading and writing raw data
 - a function to get the valid data length of a `rawbytes` variable.
-

Basic approach

This is the general approach for raw data communication. For a more detailed example of how this is done, see [Write and read rawbytes on page 141](#).

- 1 Pack data into a `rawbytes` variable (data of type `num`, `byte` or `string`).
 - 2 Write the `rawbytes` variable to a file or I/O device.
 - 3 Read a `rawbytes` variable from a file or I/O device.
 - 4 Unpack the `rawbytes` variable to `num`, `byte` or `string`.
-

Limitations

Device command communication also require the base functionality Device Command Interface and the option for the industrial network in question.

Access to files and I/O devices cannot be performed from different RAPID tasks simultaneously. Such an access is performed by all instruction in binary and character based communication, as well as `WriteRawBytes` and `ReadRawBytes`. For example, if a `ReadBin` instruction is executed in one task, then it must be ready before a `WriteRawBytes` instruction can execute in another task.

2 RobotWare-OS

2.7.3.2 RAPID components

2.7.3.2 RAPID components

Data types

This is a brief description of each data type used for raw data communication. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
<code>rawbytes</code>	<code>rawbytes</code> is used as a general data container. It can be filled with any data of types <code>num</code> , <code>byte</code> , or <code>string</code> . It also stores the length of the valid data (in bytes). <code>rawbytes</code> can contain up to 1024 bytes of data. The supported data formats are listed in the instruction <code>PackRawBytes</code> , in <i>Technical reference manual - RAPID Instructions, Functions and Data types</i> .

Instructions

This is a brief description of each instruction used for raw data communication. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
<code>ClearRawBytes</code>	<code>ClearRawBytes</code> is used to set all the contents of a <code>rawbytes</code> variable to 0. The length of the valid data in the <code>rawbytes</code> variable is set to 0. <code>ClearRawBytes</code> can also be used to clear only the last part of a <code>rawbytes</code> variable.
<code>PackRawBytes</code>	<code>PackRawBytes</code> is used to pack the contents of variables of type <code>num</code> , <code>byte</code> or <code>string</code> into a variable of type <code>rawbytes</code> .
<code>UnpackRawBytes</code>	<code>UnpackRawBytes</code> is used to unpack the contents of a variable of type <code>rawbytes</code> to variables of type <code>byte</code> , <code>num</code> or <code>string</code> .
<code>CopyRawBytes</code>	<code>CopyRawBytes</code> is used to copy all or part of the contents from one <code>rawbytes</code> variable to another.
<code>WriteRawBytes</code>	<code>WriteRawBytes</code> is used to write data of type <code>rawbytes</code> to any binary file or I/O device.
<code>ReadRawBytes</code>	<code>ReadRawBytes</code> is used to read data of type <code>rawbytes</code> from any binary file or I/O device.

Functions

This is a brief description of each function used for raw data communication. For more information, see the respective function in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
<code>RawBytesLen</code>	<code>RawBytesLen</code> is used to get the valid data length in a <code>rawbytes</code> variable.

2.7.3.3 Code examples

About the examples

These examples are simplified demonstrations of how to use `rawbytes`. For a more realistic example of how to use `rawbytes` in DeviceNet communication, see [Write rawbytes to DeviceNet on page 114](#).

Write and read rawbytes

This example shows how to pack data into a `rawbytes` variable and write it to a device. It also shows how to read and unpack a `rawbytes` variable.

```

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)
    \ISOLatin1Encoding;

  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

```

Continues on next page

2 RobotWare-OS

2.7.3.3 Code examples

Continued

```
UnpackRawBytes raw_data, 1, answer \ISOLatin1Encoding:=10;  
ENDPROC
```

Copy rawbytes

In this example, all data from raw_data_1 and raw_data_2 is copied to 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 \ISOLatin1Encoding;  
  
! 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 File and directory management

2.7.4.1 Overview

Purpose

The purpose of the file and directory management is to be able to browse and edit file structures (directories and files).

What is included

To handle file and directory management, RobotWare gives you access to:

- instructions for handling directories
 - a function for reading directories
 - instructions for handling files on a file structure level
 - functions to retrieve size and type information.
-

Basic approach

This is the general approach for file and directory management. For more detailed examples of how this is done, see [Code examples on page 145](#).

- 1 Open a directory.
- 2 Read from the directory and search until you find what you are looking for.
- 3 Close the directory.

2 RobotWare-OS

2.7.4.2 RAPID components

2.7.4.2 RAPID components

Data types

This is a brief description of each data type used for file and directory management. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
dir	dir contains a reference to a directory on disk or network. It can be linked to the physical directory with the instruction <code>OpenDir</code> .

Instructions

This is a brief description of each instruction used for file and directory management. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
<code>OpenDir</code>	<code>OpenDir</code> is used to open a directory.
<code>CloseDir</code>	<code>CloseDir</code> is used to close a directory.
<code>MakeDir</code>	<code>MakeDir</code> is used to create a new directory.
<code>RemoveDir</code>	<code>RemoveDir</code> is used to remove an empty directory.
<code>CopyFile</code>	<code>CopyFile</code> is used to make a copy of an existing file.
<code>RenameFile</code>	<code>RenameFile</code> is used to give a new name to an existing file. It can also be used to move a file from one place to another in the directory structure.
<code>RemoveFile</code>	<code>RemoveFile</code> is used to remove a file.

Functions

This is a brief description of each function used for file and directory management. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
<code>ReadDir</code>	<code>ReadDir</code> is used to retrieve the name of the next file or subdirectory under a directory that has been opened with the instruction <code>OpenDir</code> . Note that the first items read by <code>ReadDir</code> are <code>.</code> (full stop character) and <code>..</code> (double full stop characters) symbolizing the current directory and its parent directory.
<code>FileSize</code>	<code>FileSize</code> is used to retrieve the size (in bytes) of the specified file.
<code>FSSize</code>	<code>FSSize</code> (File System Size) is used to retrieve the size (in bytes) of the file system in which a specified file resides. <code>FSSize</code> can either retrieve the total size or the free size of the system.
<code>IsFile</code>	<code>IsFile</code> test if the specified file is of the specified type. It can also be used to test if the file exist at all.

2.7.4.3 Code examples

List files

This example shows how to list the files in a directory, excluding the directory itself and its parent directory (. and ..).

```

PROC lmdir(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 though 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

```

Move file to new directory

This is an example where a new directory is created, a file renamed and moved to the new directory and the old directory is removed.

```

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";

```

Continues on next page

2 RobotWare-OS

2.7.4.3 Code examples

Continued

Check sizes

In this example, the size of the file is compared with the remaining free space on the file system. If there is enough space, the file is copied.

```
VAR num freesysize;
VAR num f_size;

! Get the size of the file
f_size := FileSize("HOME:/myfile");

! Get the free size on the file system
freesysize := FSSize("HOME:/myfile" \Free);

! Copy file if enough space free
IF f_size < freesysize THEN
  CopyFile "HOME:/myfile", "HOME:/yourfile";
ENDIF
```

2.8 Fixed Position Events

2.8.1 Overview

Purpose

The purpose of Fixed Position Events is to make sure a program routine is executed when the position of the TCP is well defined.

If a move instruction is called with the zone argument set to `fine`, the next routine is always executed once the TCP has reached its target. If a move instruction is called with the zone argument set to a distance (for example `z20`), the next routine may be executed before the TCP is even close to the target. This is because there is always a delay between the execution of RAPID instructions and the robot movements.

Calling the move instruction with zone set to `fine` will slow down the movements. With Fixed Position Events, a routine can be executed when the TCP is at a specified position anywhere on the TCP path without slowing down the movement.

What is included

The RobotWare base functionality Fixed Position Events gives you access to:

- instructions used to define a position event
- instructions for moving the robot and executing the position event at the same time
- instructions for moving the robot and calling a procedure while passing the target, without first defining a position event

Basic approach

Fixed Position Events can either be used with one simplified instruction calling a procedure or it can be set up following these general steps. For more detailed examples of how this is done, see [Code examples on page 151](#).

- 1 Declare the position event.
- 2 Define the position event:
 - when it shall occur, compared to the target position
 - what it shall do
- 3 Call a move instruction that uses the position event. When the TCP is as close to the target as defined, the event will occur.

2 RobotWare-OS

2.8.2 RAPID components and system parameters

2.8.2 RAPID components and system parameters

Data types

This is a brief description of each data type in Fixed Position Events. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
triggdata	triggdata is used to store data about a position event. A position event can take the form of setting an output signal or running an interrupt routine at a specific position along the movement path of the robot. triggdata also contains information on when the action shall occur, for example when the TCP is at a defined distance from the target. triggdata is a non-value data type.
triggios	triggios is used to store data about a position event used by the instruction TriggLIOS. triggios sets the value of an output signal using a num value.
triggiosdnum	triggiosdnum is used to store data about a position event used by the instruction TriggLIOS. triggiosdnum sets the value of an output signal using a dnum value.
triggstrgo	triggstrgo is used to store data about a position event used by the instruction TriggLIOS. triggstrgo sets the value of an output signal using a stringdig value (string containing a number).

Instructions

This is a brief description of each instruction in Fixed Position Events. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
TriggIO	TriggIO defines the setting of an output signal and when to set that signal. The definition is stored in a variable of type triggdata. TriggIO can define the setting of the signal to occur at a certain distance (in mm) from the target, or a certain time from the target. It is also possible to set the signal at a defined distance or time from the starting position. By setting the distance to 0 (zero), the signal will be set when the TCP is as close to the target as it gets (the middle of the corner path).
TriggEquip	TriggEquip works like TriggIO, with the difference that TriggEquip can compensate for the internal delay of the external equipment. For example, the signal to a glue gun must be set a short time before the glue is pressed out and the gluing begins.
TriggInt	TriggInt defines when to run an interrupt routine. The definition is stored in a variable of type triggdata. TriggInt defines at what distance (in mm) from the target (or from the starting position) the interrupt routine shall be called. By setting the distance to 0 (zero), the interrupt will occur when the TCP is as close to the target as it gets (the middle of the corner path).

Continues on next page

Instruction	Description
TriggCheckIO	<p>TriggCheckIO defines a test of an input or output signal, and when to perform that test. The definition is stored in a variable of type <code>triggdata</code>.</p> <p>TriggCheckIO defines a test, comparing an input or output signal with a value. If the test fails, an interrupt routine is called. As an option the robot movement can be stopped when the interrupt occurs.</p> <p>TriggCheckIO can define the test to occur at a certain distance (in mm) from the target, or a certain time from the target. It is also possible to perform the test at a defined distance or time from the starting position.</p> <p>By setting the distance to 0 (zero), the interrupt routine will be called when the TCP is as close to the target as it gets (the middle of the corner path).</p>
TriggRampAO	<p>TriggRampAO defines the ramping up or down of an analog output signal and when this ramping is performed. The definition is stored in a variable of type <code>triggdata</code>.</p> <p>TriggRampIO defines where the ramping of the signal is to start and the length of the ramping.</p>
TriggL	<p>TriggL is a move instruction, similar to <code>MoveL</code>. In addition to the movement the <code>TriggL</code> instruction can set output signals, run interrupt routines and check input or output signals at fixed positions.</p> <p>TriggL executes up to 8 position events stored as <code>triggdata</code>. These must be defined before calling <code>TriggL</code>.</p>
TriggC	<p>TriggC is a move instruction, similar to <code>MoveC</code>. In addition to the movement the <code>TriggC</code> instruction can set output signals, run interrupt routines and check input or output signals at fixed positions.</p> <p>TriggC executes up to 8 position events stored as <code>triggdata</code>. These must be defined before calling <code>TriggC</code>.</p>
TriggJ	<p>TriggJ is a move instruction, similar to <code>MoveJ</code>. In addition to the movement the <code>TriggJ</code> instruction can set output signals, run interrupt routines and check input or output signals at fixed positions.</p> <p>TriggJ executes up to 8 position events stored as <code>triggdata</code>. These must be defined before calling <code>TriggJ</code>.</p>
TriggLIOS	<p>TriggLIOS is a move instruction, similar to <code>MoveL</code>. In addition to the movement the <code>TriggLIOS</code> instruction can set output signals at fixed positions.</p> <p>TriggLIOS is similar to the combination of <code>TriggEquip</code> and <code>TriggL</code>. The difference is that <code>TriggLIOS</code> can handle up to 50 position events stored as an array of datatype <code>triggios</code>, <code>triggiosdnum</code>, or <code>triggstrgo</code>.</p>
MoveLSync	<p>MoveLSync is a linear move instruction that calls a procedure in the middle of the corner path.</p>
MoveCSync	<p>MoveCSync is a circular move instruction that calls a procedure in the middle of the corner path.</p>
MoveJSync	<p>MoveJSync is a joint move instruction that calls a procedure in the middle of the corner path.</p>

Functions

Fixed Position Events includes no RAPID functions.

Continues on next page

2 RobotWare-OS

2.8.2 RAPID components and system parameters

Continued

System parameters

This is a brief description of each parameter in Fixed Position Events. For more information, see the respective parameter in *Technical reference manual - System parameters*.

Parameter	Description
Event Preset Time	<code>TriggEquip</code> takes advantage of the delay between the RAPID execution and the robot movement, which is about 70 ms. If the delay of the equipment is longer than 70 ms, then the delay of the robot movement can be increased by configuring <i>Event preset time</i> . <i>Event preset time</i> belongs to the type <i>Motion System</i> in the topic <i>Motion</i> .

2.8.3 Code examples

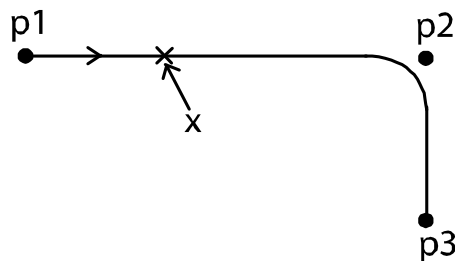
Example without Fixed Position Events

Without the use of Fixed Position Events, the code can look like this:

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

Result

The code specifies that the TCP should reach p2 before setting do1. Because the robot path is delayed compared to instruction execution, do1 is set when the TCP is at the position marked with X (see illustration).



xx0300000151

Example with TriggIO and TriggL instructions

Setting the output signal 30 mm from the target can be arranged by defining the position event and then moving the robot while the system is executing the position event.

```
VAR triggdata 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;
```

Continues on next page

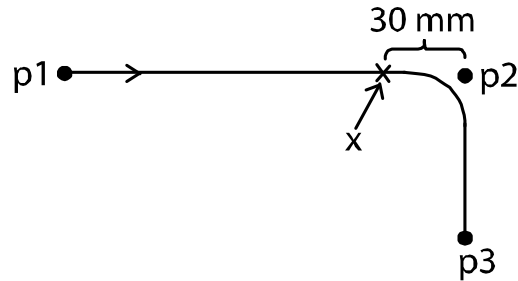
2 RobotWare-OS

2.8.3 Code examples

Continued

Result

The signal `do1` will be set when the TCP is 30 mm from `p2`. `do1` is set when the TCP is at the position marked with X (see illustration).



xx0300000158

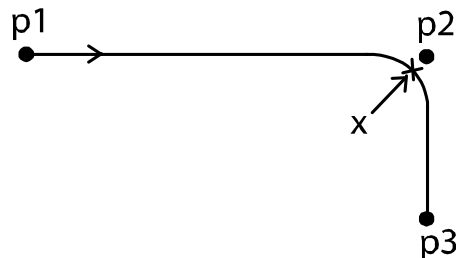
Example with MoveLSync instruction

Calling a procedure when the robot path is as close to the target as possible can be done with one instruction call.

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

Result

The procedure will be called when the TCP is at the position marked with X (see illustration).



xx0300000165

2.9 Logical Cross Connections

2.9.1 Introduction to Logical Cross Connections

Purpose

The purpose of Logical Cross Connections is to check and affect combinations of digital I/O signals (DO, DI) or group I/O signals (GO, GI). This can be used to verify or control process equipment that are external to the robot. The functionality can be compared to the one of a simple PLC.

By letting the I/O system handle logical operations with I/O signals, a lot of RAPID code execution can be avoided. Logical Cross Connections can replace the process of reading I/O signal values, calculate new values and writing the values to I/O signals.

Here are some examples of applications:

- Interrupt program execution when either of three input signals is set to 1.
- Set an output signal to 1 when both of two input signals are set to 1.

Description

Logical Cross Connections are used to define the dependencies of an I/O signal to other I/O signals. The logical operators AND, OR, and inverted signal values can be used to configure more complex dependencies.

The I/O signals that constitute the logical expression (actor I/O signals) and the I/O signal that is the result of the expression (resultant I/O signal) can be either digital I/O signals (DO, DI) or group I/O signals (GO, GI).

What is included


Logical Cross Connections allows you to build logical expressions with up to 5 actor I/O signals and the logical operations AND, OR, and inverted signal values.

2.9.2 Configuring Logical Cross Connections

System parameters

This is a brief description of the parameters for cross connections. For more information, see the respective parameter in [Configuring Logical Cross Connections on page 154](#).

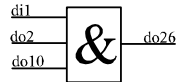
These parameters belong to the type *Cross Connection* in the topic *I/O System*.

Parameter	Description
<i>Name</i>	Specifies the name of the cross connection.
<i>Resultant</i>	The I/O signal that receive the result of the cross connection as its new value.
<i>Actor 1</i>	The first I/O signal to be used in the evaluation of the <i>Resultant</i> .
<i>Invert actor 1</i>	If <i>Invert actor 1</i> is set to <i>Yes</i> , then the inverted value of <i>Actor 1</i> is used in the evaluation of the <i>Resultant</i> .
<i>Operator 1</i>	<p>Operand between <i>Actor 1</i> and <i>Actor 2</i>. Can be either of the operands:</p> <ul style="list-style-type: none"> • AND - Results in the value 1 if both input values are 1. • OR - Results in the value 1 if at least one of the input values are 1. <p> Note The operators are calculated left to right (<i>Operator 1</i> first and <i>Operator 4</i> last).</p>
<i>Actor 2</i>	The second I/O signal (if more than one) to be used in the evaluation of the <i>Resultant</i> .
<i>Invert actor 2</i>	If <i>Invert actor 2</i> is set to <i>Yes</i> , then the inverted value of <i>Actor 2</i> is used in the evaluation of the <i>Resultant</i> .
<i>Operator 2</i>	Operand between <i>Actor 2</i> and <i>Actor 3</i> . See <i>Operator 1</i> .
<i>Actor 3</i>	The third I/O signal (if more than two) to be used in the evaluation of the <i>Resultant</i> .
<i>Invert actor 3</i>	If <i>Invert actor 3</i> is set to <i>Yes</i> , then the inverted value of <i>Actor 3</i> is used in the evaluation of the <i>Resultant</i> .
<i>Operator 3</i>	Operand between <i>Actor 3</i> and <i>Actor 4</i> . See <i>Operator 1</i> .
<i>Actor 4</i>	The fourth I/O signal (if more than three) to be used in the evaluation of the <i>Resultant</i> .
<i>Invert actor 4</i>	If <i>Invert actor 4</i> is set to <i>Yes</i> , then the inverted value of <i>Actor 4</i> is used in the evaluation of the <i>Resultant</i> .
<i>Operator 4</i>	Operand between <i>Actor 4</i> and <i>Actor 5</i> . See <i>Operator 1</i> .
<i>Actor 5</i>	The fifth I/O signal (if all five are used) to be used in the evaluation of the <i>Resultant</i> .
<i>Invert actor 5</i>	If <i>Invert actor 5</i> is set to <i>Yes</i> , then the inverted value of <i>Actor 5</i> is used in the evaluation of the <i>Resultant</i> .

2.9.3 Examples

Logical AND

The following logical structure...



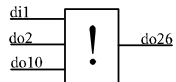
xx0300000457

... is created as shown below.

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

Logical OR

The following logical structure...



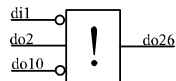
xx0300000459

... is created as shown below.

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

Inverted signals

The following logical structure (where a ring symbolize an inverted signal)...



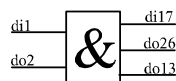
xx0300000460

... is created as shown below.

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

Several resultants

The following logical structure can not be implemented with one cross connection...



xx0300000462

Continues on next page

2 RobotWare-OS

2.9.3 Examples

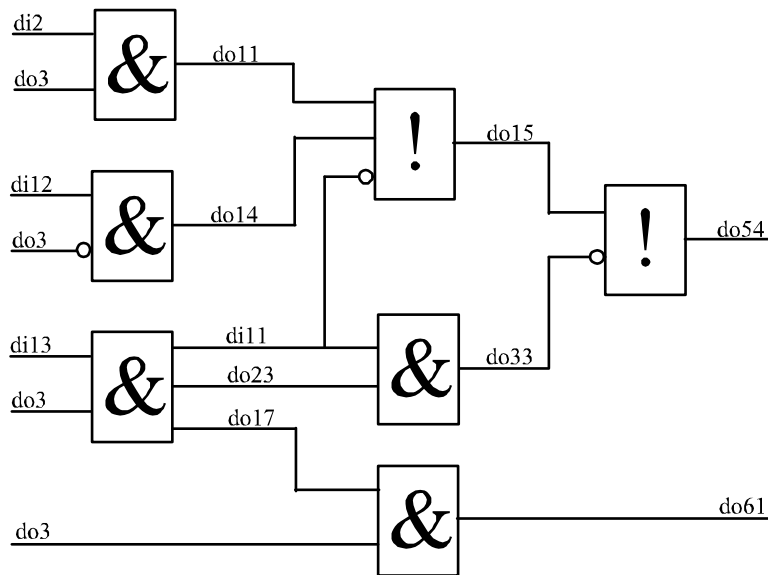
Continued

... but with three cross connections it can be implemented as shown below.

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

Complex conditions

The following logical structure...



xx0300000461

... is created as shown below.

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 Limitations

Evaluation order

If more than two actor I/O signals are used in one cross connection, the evaluation is made from left to right. This means that the operation between *Actor 1* and *Actor 2* is evaluated first and the result from that is used in the operation with *Actor 3*.

If all operators in one cross connection are of the same type (only AND or only OR) the evaluation order has no significance. However, mixing AND and OR operators, without considering the evaluation order, may give an unexpected result.

**Tip**

Use several cross connections instead of mixing AND and OR in the same cross connection.

Maximum number of actor I/O signals

A cross connection may not have more than five actor I/O signals. If more actor I/O signals are required, use several cross connections.

Maximum number of cross connections

The maximum number of cross connections handled by the robot system is 300.

Maximum depth

The maximum allowed depth of cross connection evaluations is 20.

A resultant from one cross connection can be used as an actor in another cross connection. The resultant from that cross connection can in its turn be used as an actor in the next cross connection. However, this type of chain of dependent cross connections cannot be deeper than 20 steps.

Do not create a loop

Cross connections must not form closed chains since that would cause infinite evaluation and oscillation. A closed chain appears when cross connections are interlinked so that the chain of cross connections forms a circle.

Do not have the same resultant more than once

Ambiguous resultant I/O signals are not allowed since the outcome would depend on the order of evaluation (which cannot be controlled). Ambiguous resultant I/O signals occur when the same I/O signal is resultant in several cross connections.

Overlapping device maps

The resultant I/O signal in a cross connection must not have an overlapping device map with any inverted actor I/O signals defined in the cross connection. Using I/O signals with overlapping device map in a cross connection can cause infinity signal setting loops.

2 RobotWare-OS

2.10.1 Introduction to RAPID Message Queue

2.10 RAPID Message Queue

2.10.1 Introduction to RAPID Message Queue

Purpose

The purpose of RAPID Message Queue is to communicate with another RAPID task or PC application using PC SDK.

Here are some examples of applications:

- Sending data between two RAPID tasks.
- Sending data between a RAPID task and a PC application.

RAPID Message Queue can be defined for interrupt or synchronous mode. Default setting is interrupt mode.

What is included

The RAPID Message Queue functionality is included in the RobotWare options:

- Multitasking
- RobotStudio Connect

RAPID Message Queue gives you access to RAPID instructions, functions, and data types for sending and receiving data.

Basic approach

This is the general approach for using RAPID Message Queue. For a more detailed example of how this is done, see [Code examples on page 165](#).

- 1 For *interrupt* mode: The receiver sets up a trap routine that reads a message and connects an interrupt so the trap routine is called when a new message appears.

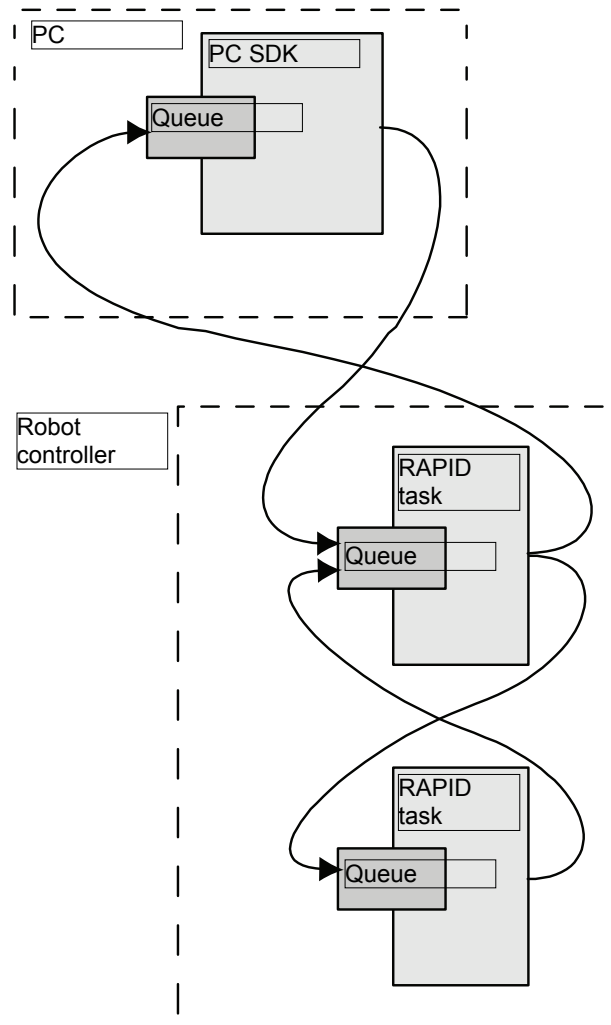
For *synchronous* mode: The message is handled by a waiting or the next executed `RMQReadWait` instruction.

- 2 The sender looks up the slot identity of the queue in the receiver task.
- 3 The sender sends the message.

2.10.2 RAPID Message Queue behavior

Illustration of communication

The picture below shows various possible senders, receivers, and queues in the system. Each arrow is an example of a way to post a message to a queue.



en070000430

Creating a PC SDK client

This manual only describes how to use RAPID Message Queue to make a RAPID task communicate with other RAPID tasks and PC SDK clients. For information about how to set up the communication on a PC SDK client, see <http://developer-center.robotstudio.com>.

What can be sent in a message

The data in a message can be any data type in RAPID, except:

- non-value
- semi-value

Continues on next page

2 RobotWare-OS

2.10.2 RAPID Message Queue behavior

Continued

- `motsetdata`

The data in a message can also be an array of a data type.

User defined records are allowed, but both sender and receiver must have identical declarations of the record.



Tip

To keep backward compatibility, do not change a user defined record once it is used in a released product. It is better to create a new record. This way, it is possible to receive messages from both old and new applications.

Queue name

The name of the queue configured for a RAPID task is the same as the name of the task with the prefix `RMQ_`, for example `RMQ_T_ROB1`. This name is used by the instruction `RMQFindSlot`.

Queue handling

Messages in queues are handled in the order that they are received. This is known as FIFO, first in first out. If a message is received while a previous message is being handled, the new message is placed in the queue. As soon as the first message handling is completed, the next message in the queue is handled.

Queue modes

The queue mode is defined with the system parameter *RMQ Mode*. Default behavior is interrupt mode.

Interrupt mode

In interrupt mode the messages are handled depending on data type. Messages are only handled for connected data types.

A cyclic interrupt must be set up for each data type that the receiver should handle. The same trap routine can be called from more than one interrupt, that is for more than one data type.

Messages of a data type with no connected interrupt will be discarded with only a warning message in the event log.

Receiving an answer to the instruction `RMQSendWait` does not result in an interrupt. No interrupt needs to be set up to receive this answer.

Synchronous mode

In synchronous mode, the task executes an `RMQReadWait` instruction to receive a message of any data type. All messages are queued and handled in order they arrive.

If there is a waiting `RMQReadWait` instruction, the message is handled immediately.

If there is no waiting `RMQReadWait` instruction, the next executed `RMQReadWait` instruction will handle the message.

Continues on next page

Message content

A RAPID Message Queue message consists of a header, containing receiver identity, and a RAPID message. The RAPID message is a pretty-printed string with data type name (and array dimensions) followed by the actual data value.

RAPID message examples:

```
"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]]"
```

RAPID task not executing

It is possible to post messages to a RAPID task queue even though the RAPID task containing the queue is not currently executing. The interrupt will not be executed until the RAPID task is executing again.

Message size limitations

Before a message is sent, the maximum size (for the specific data type and dimension) is calculated. If the size is greater than 5000 bytes, the message will be discarded and an error will be raised. The sender can get same error if the receiver is a PC SDK client with a maximum message size smaller than 400 bytes. Sending a message of a specific data type with specific dimensions will either always be possible or never possible.

When a message is received (when calling the instruction `RMQGetMsgData`), the maximum size (for the specific data type and dimension) is calculated. If the size is greater than the maximum message size configured for the queue of this task, the message will be discarded and an error will be logged. Receiving a message of a specific data type with specific dimensions will either always be possible or never possible.

Message lost

In interrupt mode, any messages that cannot be received by a RAPID task will be discarded. The message will be lost and a warning will be placed in the event log.

Example of reasons for discarding a message:

- The data type that is sent is not supported by the receiving task.
- The receiving task has not set up an interrupt for the data type that is sent, and no `RMQSendWait` instruction is waiting for this data type.
- The interrupt queue of the receiving task is full

Queue lost

The queue is cleared at power fail.

When the execution context in a RAPID task is lost, for example when the program pointer is moved to main, the corresponding queue is emptied.

Continues on next page

2 RobotWare-OS

2.10.2 RAPID Message Queue behavior

Continued

Related information

For more information on queues and messages, see *Technical reference manual - RAPID kernel*.



2.10.3 System parameters

About the system parameters

This is a brief description of each parameter in the functionality *RAPID Message Queue*. For more information, see the respective parameter in *Technical reference manual - System parameters*.

Type Task

These parameters belong to the type *Task* in the topic *Controller*.

Parameter	Description
RMQ Type	<p>Can have one of the following values:</p> <ul style="list-style-type: none"> <i>None</i> - Disable all communication with RAPID Message Queue for this RAPID task. <i>Internal</i> - Enable the receiving of RAPID Message Queue messages from other tasks on the controller, but not from external clients (FlexPendant and PC applications). The task is still able to send messages to external clients. <i>Remote</i> - Enable communication with RAPID Message Queue for this task, both with other tasks on the controller and external clients (FlexPendant and PC applications). <p>The default value is <i>None</i>.</p>
RMQ Mode	<p>Defines the mode of the queue.</p> <p>Can have one of the following values:</p> <ul style="list-style-type: none"> <i>Interrupt</i> - A message can only be received by connecting a trap routine to a specified message type. <i>Synchronous</i> - A message can only be received by executing an <code>RMQReadWait</code> instruction. <p>Default value is <i>Interrupt</i>.</p>
RMQ Max Message Size	<p>The maximum data size, in bytes, for a <i>RAPID Message Queue</i> message.</p> <p>An integer between 400 and 3000. The default value is 400.</p> <p> Note</p> <p>The value cannot be changed in RobotStudio or on the FlexPendant. The only way to change the value is to edit the <code>sys.cfg</code> file by adding the attribute <code>RmqMaxMsgSize</code> with the desired value.</p>
RMQ Max No Of Messages	<p>The maximum number of <i>RAPID Message Queue</i> messages in the queue to this task.</p> <p>An integer between 1 and 10. The default value is 5.</p> <p> Note</p> <p>The value cannot be changed in RobotStudio or on the FlexPendant. The only way to change the value is to edit the <code>sys.cfg</code> file by adding the attribute <code>RmqMaxNoOfMsg</code> with the desired value.</p>

2 RobotWare-OS

2.10.4 RAPID components

2.10.4 RAPID components

About the RAPID components

This is a brief description of each instruction, function, and data type in RAPID Message Queue. For more information, see the respective parameter in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instructions

Instruction	Description
RMQFindSlot	Find the slot identity number of the queue configured for a RAPID task or Robot Application Builder client.
RMQSendMessage	Send data to the queue configured for a RAPID task or Robot Application Builder client.
IRMQMessage	Order and enable cyclic interrupts for a specific data type.
RMQGetMessage	Get the first message from the queue of this task. Can only be used if <i>RMQ Mode</i> is defined as <i>Interrupt</i> .
RMQGetMsgHeader	Get the header part from a message.
RMQGetMsgData	Get the data part from a message.
RMQSendWait	Send a message and wait for the answer. Can only be used if <i>RMQ Mode</i> is defined as <i>Interrupt</i> .
RMQReadWait	Wait for a message. Can only be used if <i>RMQ Mode</i> is defined as <i>Synchronous</i> .
RMQEmptyQueue	Empty the queue.

Functions

Function	Description
RMQGetSlotName	Get the name of the queue configured for a RAPID task or Robot Application Builder client, given a slot identity number, i.e. given a <code>rmqslot</code> .

Data types

Data type	Description
<code>rmqslot</code>	Slot identity of a RAPID task or Robot Application Builder client.
<code>rmqmessage</code>	A message used to store data in when communicating with RAPID Message Queue. It contains information about what type of data is sent, the slot identity of the sender, and the actual data. Note: <code>rmqmessage</code> is a large data type. Declaring too many variables of this data type can lead to memory problems. Reuse the same <code>rmqmessage</code> variables as much as possible.
<code>rmqheader</code>	The <code>rmqheader</code> describes the message and can be read by the RAPID program.

2.10.5 Code examples

Example with RMQSendMessage and RMQGetMessage

This is an example where the sender creates data (x and y value) and sends it to another task. The receiving task gets the message and extract the data to the variable named data.

Sender

```

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 client

```
public void RMQReceiveRecord()
```

Continues on next page

2 RobotWare-OS

2.10.5 Code examples

Continued

```
{
  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));
}
```

Example with RMQSendWait

This is an example of a RAPID program that sends a message and wait for an answer before execution continues by getting the answer message.

```
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";
```

Continues on next page

```

! 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

```

Example with RMQReceiveSend

```

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;
            }
        }
    }
}

```

Continues on next page

2 RobotWare-OS

2.10.5 Code examples

Continued

```
        // 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));
}
```


2.11 Socket Messaging

2.11.1 Introduction to Socket Messaging

Purpose

The purpose of Socket Messaging is to allow a RAPID programmer to transmit application data between computers, using the TCP/IP network protocol. A socket represents a general communication channel, independent of the network protocol being used.

Socket communication is a standard that has its origin in Berkeley Software Distribution Unix. Besides Unix, it is supported by, for example, Microsoft Windows. With Socket Messaging, a RAPID program on a robot controller can, for example, communicate with a C/C++ program on another computer.

What is included

The RobotWare functionality Socket Messaging gives you access to RAPID data types, instructions and functions for socket communication between computers.

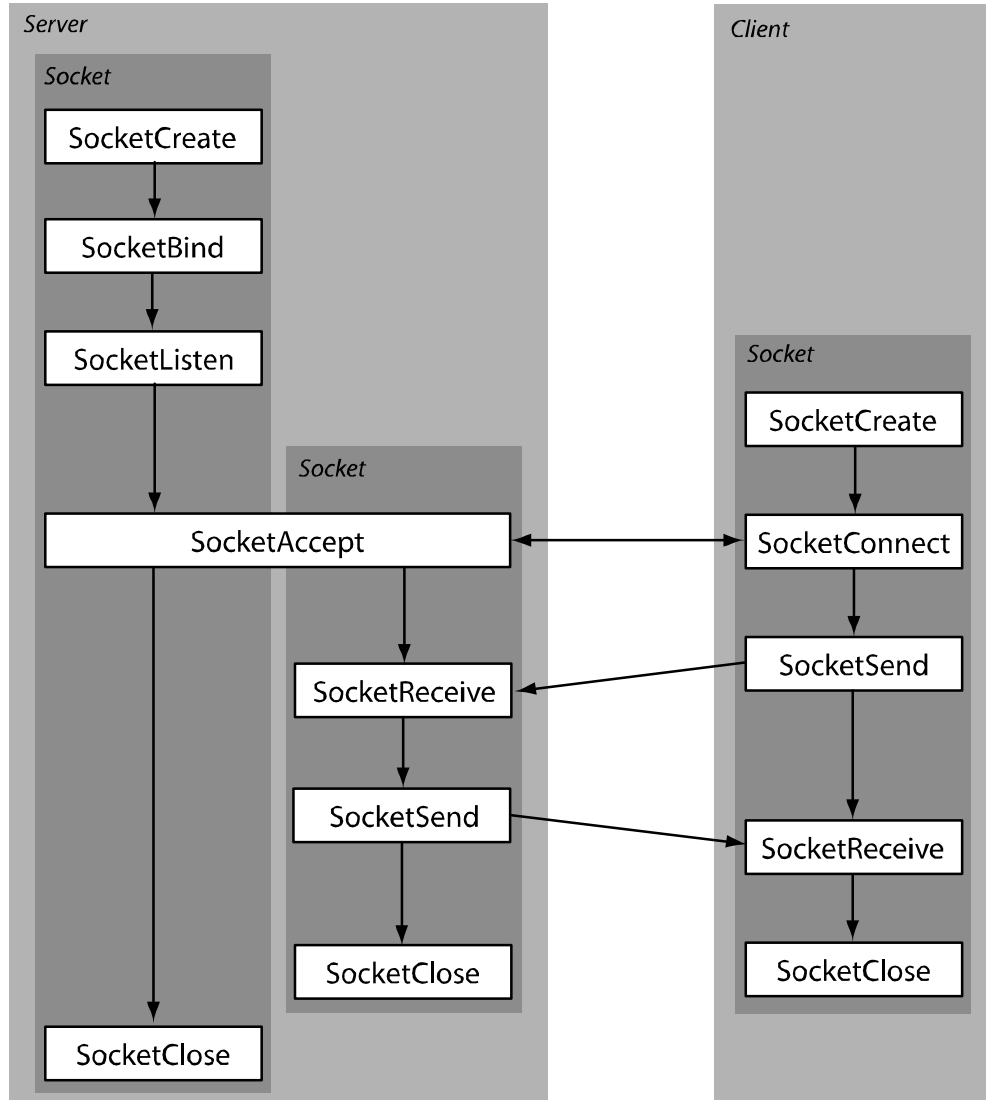
Basic approach

This is the general approach for using Socket Messaging. For a more detailed example of how this is done, see [Code examples for Socket Messaging on page 174](#).

- 1 Create a socket, both on client and server. A robot controller can be either client or server.
- 2 Use `SocketBind` and `SocketListen` on the server, to prepare it for a connection request.
- 3 Order the server to accept incoming socket connection requests.
- 4 Request socket connection from the client.
- 5 Send and receive data between client and server.

2.11.2 Schematic picture of socket communication

Illustration of socket communication



en0600003224



Tip

Do not create and close sockets more than necessary. Keep the socket open until the communication is completed. The socket is not really closed until a certain time after `SocketClose` (due to TCP/IP functionality).

2.11.3 Technical facts about Socket Messaging

Overview

When using the functionality Socket Messaging to communicate with a client or server that is not a RAPID task, the following information can be useful.

No string termination

When sending a data message, no string termination sign is sent in the message. The number of bytes sent is equal to the return value of the function `strlen(str)` in the programming language C.

Unintended merge of messages

If sending two messages with no delay between them, the result can be that the second message is appended to the first. The result is one big message instead of two messages. To avoid this, use acknowledge messages from the receiver of the data, if the client/server is just receiving messages.

Non printable characters

If a client that is not a RAPID task needs to receive non printable characters (binary data) in a string from a RAPID task, this can be done by RAPID as shown in the example below.

```
SocketSend socket1 \Str:="\0D\0A";
```

For more information, see *Technical reference manual - RAPID kernel*, section *String literals*.

2 RobotWare-OS

2.11.4 RAPID components

2.11.4 RAPID components

Data types

This is a brief description of each data type in Socket Messaging. For more information, see *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
socketdev	A socket device used to communicate with other computers on a network.
socketstatus	Can contain status information from a <code>socketdev</code> variable.

Instructions for client

This is a brief description of each instruction used by the a Socket Messaging client. For more information, see *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
SocketCreate	Creates a new socket and assigns it to a <code>socketdev</code> variable.
SocketConnect	Makes a connection request to a remote computer. Used by the client to connect to the server.
SocketSend	Sends data via a socket connection to a remote computer. The data can be a <code>string</code> or <code>rawbytes</code> variable, or a <code>byte</code> array.
SocketReceive	Receives data and stores it in a <code>string</code> or <code>rawbytes</code> variable, or in a <code>byte</code> array.
SocketClose	Closes a socket and release all resources.



Tip

Do not use `SocketClose` directly after `SocketSend`. Wait for acknowledgement before closing the socket.

Instructions for server

A Socket Messaging server uses the same instructions as the client, except for `SocketConnect`. In addition, the server use the following instructions:

Instruction	Description
SocketBind	Binds the socket to a specified port number on the server. Used by the server to define on which port (on the server) to listen for a connection. The IP address defines a physical computer and the port defines a logical channel to a program on that computer.
SocketListen	Makes the computer act as a server and accept incoming connections. It will listen for a connection on the port specified by <code>SocketBind</code> .
SocketAccept	Accepts an incoming connection request. Used by the server to accept the client's request.

Continues on next page



Note

The server application must be started before the client application, so that the instruction `SocketAccept` is executed before any client execute `SocketConnect`.

Functions

This is a brief description of each function in Socket Messaging. For more information, see *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
<code>SocketGetStatus</code>	Returns information about the last instruction performed on the socket (created, connected, bound, listening, closed). <code>SocketGetStatus</code> does not detect changes from outside RAPID (such as a broken connection).

2.11.5 Code examples for Socket Messaging

Example of client/server communication

This example shows program code for a client and a server, communicating with each other.

The server will write on the FlexPendant:

```
Client wrote - Hello server
Client wrote - Shutdown connection
```

The client will write on its FlexPendant:

```
Server wrote - Message acknowledged
Server wrote - Shutdown acknowledged
```

In this example, both the client and the server use RAPID programs. In reality, one of the programs would often be running on a PC (or similar computer) and be written in another program language.

Code example for client, contacting server with IP address 192.168.0.2:

```
! 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
```

Code example for server (with IP address 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;
```

Continues on next page

```

! Communication
SocketReceive client_socket \Str:=received_string;
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

```

Example of error handler

The following error handlers will take care of power failure or broken connection.

Error handler for client in previous example:

```

! Error handler to make it possible to handle power fail
ERROR
IF ERRNO=ERR_SOCK_TIMEOUT THEN
  RETRY;
ELSEIF ERRNO=ERR_SOCK_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

```

Error handler for server in previous example:

```

! Error handler for power fail and connection lost
ERROR
IF ERRNO=ERR_SOCK_TIMEOUT THEN
  RETRY;
ELSEIF ERRNO=ERR_SOCK_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

```

2 RobotWare-OS

2.12.1 Introduction to User logs

2.12 User logs

2.12.1 Introduction to User logs

Description

The RobotWare base functionality *User logs* generates event logs for the most common user actions. The event logs are generated in the group *Operational events*, number series *1 xxxx*.

For more information on handling the event log, see *Operating manual - OmniCore* and *Technical reference manual - Event logs for RobotWare 7*.

Purpose

The purpose of *User logs* is to track changes in the robot controller related to user actions. This can for example be helpful to find the root cause if a production stop occurs.

What is included

The RobotWare base functionality *User logs* generates event logs for the following changes related to user actions:

Topic	User action
Program execution	Changing the speed or run mode (single cycle/continuous). Making changes to the task selection panel. Setting or resetting non motion execution mode.
Simulate wait instructions	Simulating wait instructions, for example <code>WaitTime</code> , <code>WaitUntil</code> , <code>WaitDx</code> , etc.
RAPID changes	Opening or closing RAPID programs or modules, editing RAPID code, or modifying robot positions.
Program pointer movements	Moving the program pointer to main, to a routine, to a position, or to a service routine (call routine).
Changes on the mechanical unit	Updating the revolution counters or performing a calibration.
Jogging	Changing the tool, the work object, the payload, the coordinate system, or go to a position.
Supervision	Setting or resetting the jog or path supervision. Setting the level of supervision.
Change of configuration	Loading configuration data or changing a configuration attribute.
System changes	Clearing the event log or changing date and time.
Serial measurement board	Changing the data in the serial measurement board or changing the data in the robot memory.

3 Motion Performance

3.1 Absolute Accuracy [3101-x]

3.1.1 About Absolute Accuracy

Purpose

Absolute Accuracy is a calibration concept that improves TCP accuracy. The difference between an ideal robot and a real robot can be several millimeters, resulting from mechanical tolerances and deflection in the robot structure. *Absolute Accuracy* compensates for these differences.

Here are some examples of when this accuracy is important:

- Exchangeability of robots
- Offline programming with no or minimum touch-up
- Online programming with accurate movement and reorientation of tool
- Programming with accurate offset movement in relation to eg. vision system or offset programming
- Re-use of programs between applications

The option *Absolute Accuracy* is integrated in the controller algorithms and does not need external equipment or calculation.



Note

The performance data is applicable to the corresponding RobotWare version of the individual robot.



Note

Singularities might appear in slightly different positions on a real robot compared to RobotStudio, where *Absolute Accuracy* is off compared to the real controller.

What is included

Every *Absolute Accuracy* robot is delivered with:

- compensation parameters saved in the robot memory
- a birth certificate representing the *Absolute Accuracy* measurement protocol for the calibration and verification sequence.

A robot with *Absolute Accuracy* calibration has a label with this information on the manipulator.

Absolute Accuracy supports floor mounted, wall mounted, and ceiling mounted installations. The compensation parameters that are saved in the robot memory differ depending on which *Absolute Accuracy* option is selected.

Continues on next page

3 Motion Performance

3.1.1 About Absolute Accuracy

Continued

When is *Absolute Accuracy* being used

Absolute Accuracy works on a robot target in Cartesian coordinates, not on the individual joints. Therefore, joint based movements (e.g. `MoveAbsJ`) will not be affected.

If the robot is inverted, the Absolute Accuracy calibration must be performed when the robot is inverted.

Absolute Accuracy active

Absolute Accuracy will be active in the following cases:

- Any motion function based on robtargets (e.g. `MoveL`) and ModPos on robtargets
- Reorientation jogging
- Linear jogging
- Tool definition (4, 5, 6 point tool definition, room fixed TCP, stationary tool)
- Work object definition

Absolute Accuracy not active

The following are examples of when Absolute Accuracy is not active:

- Any motion function based on a jointtarget (`MoveAbsJ`)
- Independent joint
- Joint based jogging
- Additional axes
- Track motion



Note

In a robot system with, for example, an additional axis or track motion, the Absolute Accuracy is active for the manipulator but not for the additional axis or track motion.

RAPID instructions

There are no RAPID instructions included in this option.

3.1.2 Useful tools

Overview

The following products are recommended for operation and maintenance of Absolute Accurate robots:

- Load Identification
- CalibWare (Absolute Accuracy calibration tool)

Load Identification

Absolute Accuracy calculates the robot's deflection depending on payload. It is very important to have an accurate description of the load.

Load Identification is a tool that determines the mass, center of gravity, and inertia of the payload.

For more information, see *Operating manual - OmniCore*.

CalibWare

CalibWare, provided by ABB, is a tool for calibrating Absolute Accuracy. The documentation to CalibWare describes the Absolute Accuracy calibration procedure in detail.

CalibWare is used at initial calibration and when servicing the robot.

3 Motion Performance

3.1.3 Configuration

3.1.3 Configuration

Activate Absolute Accuracy

Use RobotStudio and follow these steps (see *Operating manual - RobotStudio* for more information):

- 1 If you do not already have write access, click **Request Write Access** and wait for grant from the FlexPendant.
- 2 Click **Configuration Editor** and select **Motion**.
- 3 Click the type **Robot**.
- 4 For the parameter *Use Robot Calibration*, change the value to *r1_calib*.
- 5 No restart is required.

Deactivate Absolute Accuracy

Use RobotStudio and follow these steps (see *Operating manual - RobotStudio* for more information):

- 1 If you do not already have write access, click **Request Write Access** and wait for grant from the FlexPendant.
- 2 Click **Configuration Editor** and select the topic **Motion**.
- 3 Click the type **Robot**.
- 4 Configure the parameter *Use Robot Calibration* and change the value to "r1_uncalib".
- 5 No restart is required.

Change calibration data

If you exchange the manipulator, the calibration data for the new manipulator must be loaded. This is done by copying the calibration data from the robot memory to the robot controller.

Use the FlexPendant and follow these steps (for more information, see *Operating manual - OmniCore*):

- 1 On the start screen, tap **Calibrate**, and then select **Calibration** from the menu.
- 2 Tap on the robot you wish to update.
- 3 Tap the tab **Robot Memory**.
- 4 Tap **Advanced**.
- 5 Tap **Clear Controller Memory**.
- 6 Tap **Clear** and then confirm by tapping **Yes**.
- 7 Tap **Close**.
- 8 Tap **Update**.
- 9 Tap **Cabinet or robot has been exchanged** and confirm by tapping **Yes**.

3.1.4 Maintenance

3.1.4.1 Maintenance that affect the accuracy

Overview

This section will focus on those maintenance activities that directly affect the accuracy of the robot, summarized as follows:

- Tool recalibration
- Motor replacement
- Wrist replacement (large robots)
- Arm replacement (lower arm, upper arm, gearbox, foot)
- Manipulator replacement
- Loss of accuracy



Note

If the RobotWare version on the controller must be downgraded, then contact your local ABB for support regarding compatible versions of Absolute Accuracy.

Tool recalibration

For information about tool recalibration, see [Tool calibration on page 195](#).

Motor replacement

Replacement of all motors requires a re-calibration of the corresponding resolver offset parameter using the standard calibration method for the respective robot. This is described in the product manual for the robot.

If the motor replacement requires disassembly of the arm, then see [Arm replacement or disassembly on page 181](#).

Wrist replacement

Replacement of the wrist unit requires a re-calibration of the resolver offsets for axes 5 and 6 using the standard calibration method for the respective robot.

Arm replacement or disassembly

Replacement of any of the robot arms, or other mechanical structure (excluding wrist), changes the structure of the robot to the extent that a robot recalibration is required. It is recommended that, after an arm replacement, the entire robot should be recalibrated to ensure optimal Absolute Accuracy functionality. This is typically performed with CalibWare and a separate measurement system. CalibWare can be used together with any generic 3Dmeasurement system.

For more information about the calibration process, see documentation for CalibWare.


Continues on next page

3 Motion Performance

3.1.4.1 Maintenance that affect the accuracy

Continued

A summary of the calibration process is presented as follows:

	Action
1	Replace the affected component.
2	Perform a resolver offset calibration for all axes. See the product manual for the respective robot.
3	Recalibrate the TCP.
4	Check the accuracy by comparison to a fixed reference point in the cell.
5	Check the accuracy of the work objects.  Note An update of the defined work objects will make the deviation less in positioning.
6	Check the accuracy of the positions in the current application.
7	If the accuracy still is unsatisfactory, perform an Absolute Accuracy calibration of the entire robot. See documentation for CalibWare.

Manipulator replacement

When a robot manipulator is replaced without replacing the controller cabinet, it is necessary to update the Absolute Accuracy parameters in the controller cabinet and realign the robot to the cell. The Absolute Accuracy parameters are updated by loading the replacement robot's calibration parameters into the controller as described in [Change calibration data on page 180](#). Ensure that the calibration data is loaded and that Absolute Accuracy is activated.

The alignment of the replacement robot to the cell depends on the robot alignment technique chosen at installation. If the robot mounting pins are aligned to the cell then the robot need only be placed on the pins - no further alignment is necessary. If the robot was aligned using a robot program then it is necessary to measure the cell fixture(s) and measure the robot in several positions (for best results use the same program as the original robot). See [Measure robot alignment on page 193](#).

3.1.4.2 Loss of accuracy

Cause and action

Loss of accuracy usually occur after robot collision or large temperature variations. It is necessary to determine the cause of the errors, and take adequate action.

If...	...then...
the tool is not properly calibrated	recalibrate if the TCP has changed.
the tool load is not correctly defined	run Load Identification to ensure correct mass, centre of gravity and inertia for the active tool.
the resolver offsets are no longer valid	<ol style="list-style-type: none"> 1 Check that the axis scales show that the robot stands correctly in the home position. 2 If the indicators are not aligned, move the robot to correct position and update the revolution counters. 3 If the indicators are close to aligned but not correct, re-calibrate with the standard calibration for the robot.
the robot's relationship to the fixture(s) has changed	<ol style="list-style-type: none"> 1 Check by moving the robot to a predefined position on the fixture(s). 2 Visually assessing whether the deviation is excessive. 3 If excessive, realign robot to fixture(s).
the robot structure has changed	<ol style="list-style-type: none"> 1 Visually assess whether the robot is damaged. 2 If damaged then replace entire manipulator -or- replace affected arm(s) -or- recalibrate affected arm(s).

3 Motion Performance

3.1.5.1 Error sources

3.1.5 Compensation theory

3.1.5.1 Error sources

Types of errors

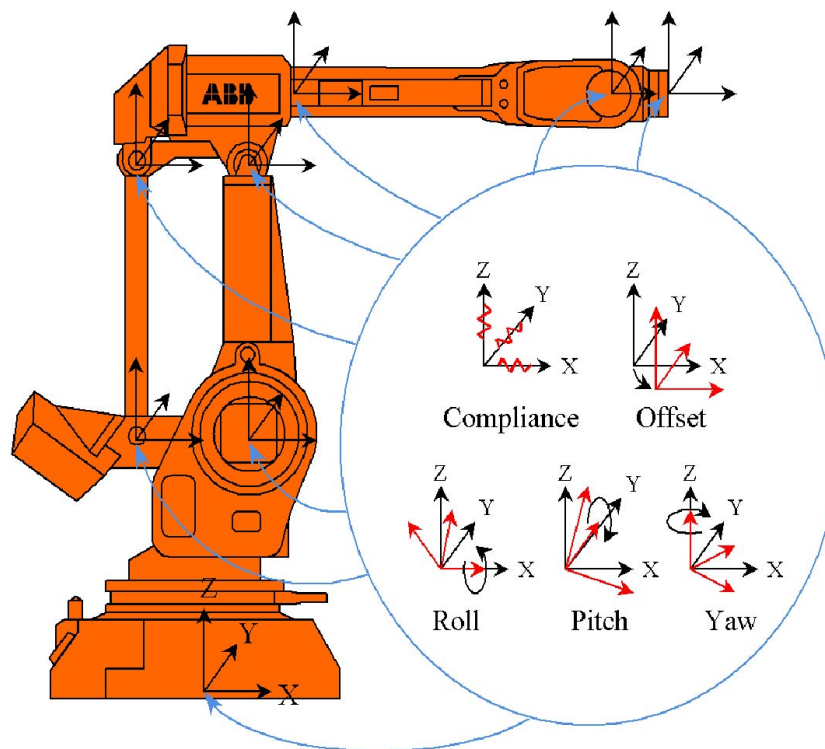
The errors compensated for in the controller derive from the mechanical tolerances of the constituent robot parts. A subset of these are detailed in the illustration below.

Compliance errors are due to the effect of the robot's own weight together with the weight of the current payload. These errors depend on gravity and the characteristics of the load. The compensation of these errors is most efficient if you use Load Identification (see *Operating manual - OmniCore*).

Kinematic errors are caused by position or orientational deviations in the robot axes. These are independent of the load.

Illustration

There are several types of errors that can occur in each joint.



en030000232

3.1.5.2 Absolute Accuracy compensation

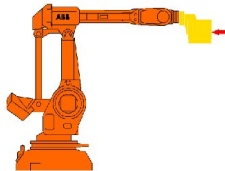
Introduction

Both compliance and kinematic errors are compensated for with "fake targets". Knowing the deflection of the robot (i.e. deviation from ordered position), *Absolute Accuracy* can compensate by ordering the robot to a fake target.

The compensation works on a robot target in cartesian coordinates, not on the individual joints. This means that it is the position of the TCP (marked with an arrow in the following illustrations) that is correctly compensated.

Desired position

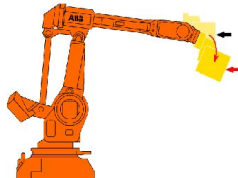
The following illustration shows the position you want the robot to have.



xx0300000225

Position due to deflection

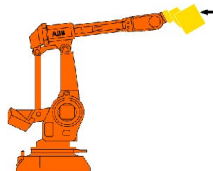
The following illustration shows the position the robot will get without *Absolute Accuracy*. The weight of the robot arms and the load will make a deflection on the robot. Note that the deflection is exaggerated.



xx0300000227

Fake target

In order to get the desired position, *Absolute Accuracy* calculates a fake target. When you enter a desired position, the system recalculates it to a fake target that after the deflection will result in the desired position.



xx0300000226

Continues on next page

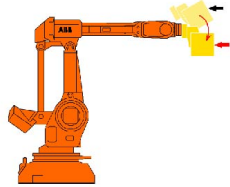
3 Motion Performance

3.1.5.2 Absolute Accuracy compensation

Continued

Compensated position

The actual position will be the same as your desired position. As a user you will not notice the fake target or the deflection. The robot will behave as if it had no deflection.



xx0300000224

3.1.6 Preparation of Absolute Accuracy robot

3.1.6.1 ABB calibration process

Overview

This section describes the calibration process that ABB performs on each Absolute Accuracy robot, regardless of robot type or family, before it is delivered.

The process can be divided in four steps:

- 1 Resolver offset calibration
- 2 Absolute Accuracy calibration
- 3 Calibration data stored in the robot memory
- 4 Absolute Accuracy verification
- 5 Generation of a birth certificate

Resolver offset calibration

The resolver offset calibration process is used to calibrate the resolver offset parameters.

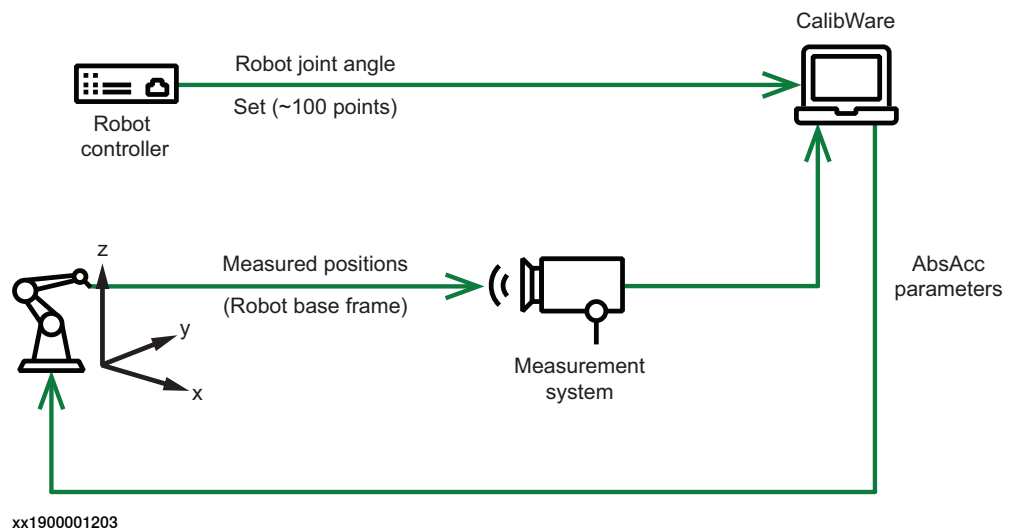
For information on how to do this, see the product manual for the respective robot.

Absolute Accuracy calibration

The Absolute Accuracy calibration is performed on top of the resolver offset calibration, hence the importance of having repeatable methods for both processes.

Each robot is calibrated with maximum load to ensure that the correct compensation parameters are detected (calibration at lower load might not result in a correct determination of the robot flexibility parameters.) The process runs the robot to 100 jointtarget poses and measures each corresponding measurement point coordinate. The list of poses and measurements are fed into the CalibWare calibration core and a set of robot compensation parameters are created.

For information on how to do this, see documentation for CalibWare.



Continues on next page

3 Motion Performance

3.1.6.1 ABB calibration process

Continued

Absolute Accuracy verification

The parameters are loaded onto the controller and activated. The robot is then run to a set of 50 robot target poses. Each pose is measured and the deviation from nominal determined.

For information on how to do this, see documentation for CalibWare.

The requirements for acceptance vary between robot types, see typical performance data in the product specification for the respective robot.

Compensation parameters and birth certificate

The compensation parameters are saved in the robot memory (see [Compensation parameters on page 190](#)).

A birth certificate is created representing the Absolute Accuracy measurement protocol for the calibration and verification sequence (see [Birth certificate on page 189](#)).

3.1.6.2 Birth certificate

About the birth certificate

All Absolute Accuracy robots are shipped with a birth certificate. It represents the Absolute Accuracy measurement protocol for the calibration and verification sequence.

The birth certificate contains the following information:

- Robot information (robot type, serial number, version of Absolute Accuracy)
- Accuracy information (maximum, average and standard deviation for finepoint error distribution)
- Tool information (TCP, mass, center of gravity)
- Description of measurement protocol (measurement and calibration system, number of points, measurement point location)

3 Motion Performance

3.1.6.3 Compensation parameters

3.1.6.3 Compensation parameters

About the compensation parameters

All Absolute Accuracy robots are shipped with a set of compensation parameters, as part of the system parameters (configuration). As the resolver offset calibration is integral in the Absolute Accuracy calibration, the resolver offset parameters are also stored in the robot memory.

The compensation parameters

The compensation parameters are defined in the following configuration types:

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

The type ROBOT_CALIB defines the top level of the calibration structure. The instance *r1_calib* activates the Absolute Accuracy functionality by specifying the flag *-absacc*. See [Activate Absolute Accuracy on page 180](#).

The types ARM_CALIB, JOINT_CALIB, PARALLEL_ARM_CALIB, and MOTOR_CALIB are reserved by the system and are only shown when the Absolute Accuracy option is selected in the **Modify Installation** dialog. The parameter values can be changed by importing a new configuration file.

The compensation parameters are included in a backup, in the file *moc.cfg*.

3.1.7 Cell alignment

3.1.7.1 Overview

About cell alignment

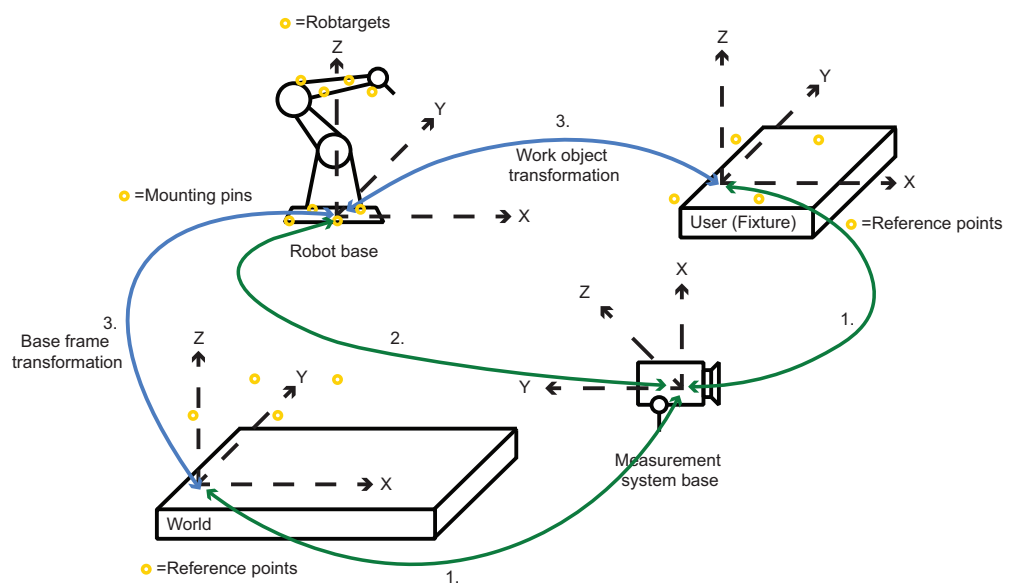
The compensation parameters for the Absolute Accuracy robot are determined from the physical base plate to the robot tool. For many applications this is enough, the robot can be used as any other robot. However, it is common that Absolute Accuracy robots are aligned to the coordinates in their cells. This section describes this alignment procedure. For a more detailed description, see documentation for CalibWare.

Alignment procedure

In order for the robot to be accurate with respect to the entire robot cell, it is necessary to install the robot correctly. In summary, this involves:

	Action	Description
1	Measure fixture alignment	Determine the relationship between the measurement system and the fixture. See Measure fixture alignment on page 192 .
2	Measure robot alignment	Determine the relationship between the measurement system and the robot. See Measure robot alignment on page 193 .
3	Calculate frame relationships	Determine the relationship between, for example, the robot and the fixture. See Frame relationships on page 194 .
4	Calibrate tool	Determine the relationship between the robot tool and other cell components. See Tool calibration on page 195 .

Illustration



en0300000239

3 Motion Performance

3.1.7.2 Measure fixture alignment

3.1.7.2 Measure fixture alignment

About fixture alignment

A fixture is defined as a cell component that is associated with a particular coordinate system. The interaction between the robot and the fixture requires an accurate relationship in order to ensure Absolute Accuracy.

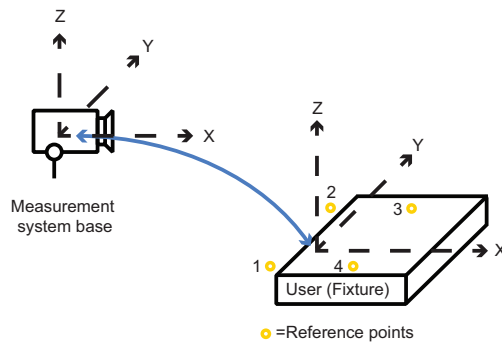
Absolute Accuracy fixtures must be equipped with at least three (preferably four) reference points, each with clearly marked position information.

Fixture measurement procedure

The alignment of the fixture is done in the following steps:

- 1 Enter the reference point names and positions into the alignment software.
- 2 Measure the reference points and assign the same names.
- 3 Use the alignment software to match the reference to measured points and determine the relationship frame. All measurement systems support this form of transformation.

Illustration



en0300000237

Measurement positions	Reference positions	Frame relationship
Pos1: 100, 100, 200	Pos1: 100, 100, 100	1) RobotStudio work object (0,0,-100,0,0,0) (x,y,z,roll,pitch,yaw)
Pos2: 100, 200, 200	Pos2: 100, 200, 100	
Pos3: 200, 200, 200	Pos3: 200, 200, 100	
Pos4: 200, 100, 200	Pos4: 200, 100, 100	

3.1.7.3 Measure robot alignment

Select method

The relationship between the measurement system and the robot can be determined in the following ways:

Alignment procedure	Description
Alignment to physical base	The equivalent to the fixture alignment in which the physical base pins are measured and aligned with respect to the reference positions detailed in the product manual for the respective robot.
Alignment to theoretical base	Measuring several robot poses and letting the alignment software determine the robot alignment.

Alignment to physical base

The advantage of aligning the robot as a fixture is in its simplicity - the robot is treated as another fixture in the cell and its base points measured accordingly. The disadvantage is that small errors in the subsequent placement of the robot on the pins can result in large TCP errors due to the reach of the robot (i.e. the placement of the robot is not calibrated.)

In order to determine the reference point coordinates, it is necessary to consult the product manual for that robot type.

Once the correct point has been measured, the alignment software is used to determine the frame relationship between the measurement system and robot base.

Alignment to theoretical base

The advantage of aligning the robot to a theoretical base is that any errors resulting from mounting the robot can be eliminated. Furthermore, the alignment process details the robot accuracy at the measured points, confirming correct Absolute Accuracy functionality. The disadvantage is that a robot program must be created (either manually or automatically from CalibWare) and the robot measured (ideally with correct tool however the TCP can also be calibrated as a part of this procedure.)

Once the correct point is measured, the alignment software is used to determine the frame relationship between the measurement system and robot base.

3 Motion Performance

3.1.7.4 Frame relationships

3.1.7.4 Frame relationships

About frame relationships

Once the relationships between the measurement system and all other cell components are measured, the relationships between cell components can be determined.

The relationship between the world coordinate system and the robot shall be stored in the robot base. The relationship between the robot and the fixture shall be stored in the `workobject` data type.

The measurement system is initially the active coordinate system as both world and robot are measured relative to the measurement system.

Determine robot base

Use a standard measurement system software to determine the robot base in world coordinates:

- 1 Set the world coordinate system to be active (the origin).
- 2 Read the coordinates of the robot base frame (now relative to the world).

The fixture relationship is similarly determined by setting the robot to be active and reading the coordinates of the fixture frame.

3.1.7.5 Tool calibration

About tool calibration

The Absolute Accuracy robot compensation parameters are calculated to be tool independent. This allows any tool with a correctly pre-defined TCP to be connected to the robot flange and used without requiring a tool re-calibration. In practice, however, it is difficult to perform a correct TCP calibration with, for example, a Coordinate Measurement Machine (CMM) as this does not take into account the connection of the tool to the robot nor the tool flexibility.

Each tool should be calibrated on a regular basis to ensure optimal robot accuracy.

Tool calibration procedures

Suggested tool recalibration procedures are detailed as follows:

- SBCU (Single Beam Calibration Unit) such as the ABB BullsEye for arc-welding or spot-welding applications.
- Geometry calibration such as the 4, 5 or 6 Point tool center point calibration routine available in the controller. A measurement system can be used to ensure that the single point used is accurate.
- RAPID tool calibration routines: MToolTCPCalib (calibration of TCP for moving tool), SToolTCPCalib (calibration of TCP for stationary tool), MToolRotCalib (calibration of rotation for moving tool), SToolRotCalib (calibration of TCP and rotation for stationary tool.)
- Using theoretical data, for example from a CAD model.



Tip

As the tool load characteristics are used in the Absolute Accuracy models, it is essential that all parameters be as accurate as possible. Use of Load Identification is an efficient method of determining tool load characteristics.

3 Motion Performance

3.2.1 About Ultra Accuracy

3.2 Ultra Accuracy [3101-10]

3.2.1 About Ultra Accuracy

Purpose

Ultra Accuracy is a control concept for GoFa CRB 15000 robots that improves the TCP path accuracy even further.

Here are some examples of when this kind of accuracy is important:

- Gluing extremely small electronics components.
- Laser welding of car body parts in automotive.
- Positioning of composite material layup in aircraft manufacturing.
- High fidelity 3D printing.
- High precision laser cutting of small metal shapes.

The option *Ultra Accuracy* is integrated in the controller algorithms and does not need external equipment.



Note

The performance data is applicable to the corresponding RobotWare version of the individual robot.

What is included

A robot with the option *Ultra Accuracy* is delivered with:

- compensation parameters saved in the robot memory
- a birth certificate representing the *Ultra Accuracy* measurement protocol for the calibration and verification sequence.

A robot with *Ultra Accuracy* capability has a label with this information on the manipulator.

Ultra Accuracy supports floor mounted, wall mounted, and ceiling mounted installations.

When is *Ultra Accuracy* being used

Ultra Accuracy works on a all robot target in Cartesian coordinates and also on the individual joints. Therefore, joint based movements, for example, `MoveAbsJ`, are also affected.

Ultra Accuracy active

Ultra Accuracy is active in the following cases:

- Any motion function based on `robtarget` or `jointtarget` (for example, `MoveL` and `MoveAbsJ`) and `ModPos` on `robtargets`.
- Reorientation jogging
- Linear jogging
- Tool definition (4, 5, 6 point tool definition, room fixed TCP, stationary tool)
- Work object definition

Continues on next page

Ultra Accuracy not active

The following are examples of when *Ultra Accuracy* is not active:

- Independent joint
- Applications such as Force Control and SoftMove

RAPID instructions

There are no RAPID instructions included in this option.

3 Motion Performance

3.2.2 Configuration

3.2.2 Configuration

Overview

Use RobotStudio to activate and deactivate *Ultra Accuracy* or to change the stability margin.

See *Operating manual - RobotStudio*.

Activate *Ultra Accuracy*

- 1 On the **Controller** tab, click **Request Write Access** and wait for grant from the FlexPendant.
- 2 Click **Configuration Editor** and select *Motion*.
- 3 Click the type *Robot*.
- 4 For the parameter *Ultra Accuracy*, change the value to *Active*.

No restart of the controller is needed.

Deactivate *Ultra Accuracy*

- 1 On the **Controller** tab, click **Request Write Access** and wait for grant from the FlexPendant.
- 2 Click **Configuration Editor** and select the topic *Motion*.
- 3 Click the type *Robot*.
- 4 For the parameter *Ultra Accuracy*, change the value to *Inactive*.

No restart of the controller is needed.

Change the stability margin

If vibrations are seen on the robot there might be a need to change the stability margin. This can be the case if a special tool is used, etc.

- 1 On the **Controller** tab, click **Request Write Access** and wait for grant from the FlexPendant.
- 2 Click **Configuration Editor** and select the topic *Motion*.
- 3 Click the type *Arm*.
- 4 Select the arm that needs a lower stability margin for *Ultra Accuracy*.
- 5 For the parameter *Ultra Accuracy stability margin*, change the value.
Recommended step size is 0.1.

No restart of the controller is needed.

3.2.3 Compensation parameters

About the compensation parameters

All *Ultra Accuracy* robots are shipped with a set of compensation parameters apart from the *Absolute Accuracy* parameters, as part of the system parameters (configuration). These are identified with a service routine.

The compensation parameters

The compensation parameters are defined in the following configuration types:

- HD_KINEMATIC_ERROR_COMP
- SECONDARY_ENCODER_ERROR_COMP
- DISTRIBUTED_FRICTION_COMP

They are all automatically created and should not be manually changed.

The compensation parameters are included in a backup, in the file moc.cfg and are also stored in robot memory.

3 Motion Performance

3.3 Advanced Robot Motion 3100-1

3.3 Advanced Robot Motion 3100-1

About Advanced Robot Motion

The option *Advanced Robot Motion* gives you access to:

- *Advanced Shape Tuning*, see [Advanced Shape Tuning \[included in 3100-1\] on page 201](#).
- Changing *Motion Process Mode* from RAPID, see [Motion Process Mode \[included in 3100-1\] on page 209](#).
- *Wrist Move*, see [Wrist Move \[included in 3100-1\] on page 217](#).

3.4 Advanced Shape Tuning [included in 3100-1]

3.4.1 About Advanced Shape Tuning

Purpose

The purpose of *Advanced Shape Tuning* is to reduce the path deviation caused by joint friction of the robot.

Advanced Shape Tuning is useful for low speed cutting (10-100 mm/s) of, for example, small circles. Effects of robot joint friction can cause path deviation of typically 0.5 mm in these cases. By tuning parameters of a friction model in the controller, the path deviation can be reduced to the repeatability level of the robot, for example, 0.1 mm for a medium sized robot.

What is included

Advanced Shape Tuning is included in the RobotWare option *Advanced robot motion* and gives you access to:

- Instructions `FricIdInit`, `FricIdEvaluate` and `FricIdSetFricLevels` that automatically optimize the joint friction model parameters for a programmed path.
- The system parameters *Friction FFW On*, *Friction FFW level* and *Friction FFW Ramp* for manual tuning of the joint friction parameters.
- The tune types `tune_fric_lev` and `tune_fric_ramp` that can be used with the instruction `TuneServo`.

Basic approach

This is a brief description of how *Advanced Shape Tuning* is most commonly used:

- 1 Set system parameter *Friction FFW On* to TRUE. See [System parameters on page 206](#).
- 2 Perform automatic tuning of the joint friction levels using the instructions `FricIdInit` and `FricIdEvaluate`. See [Automatic friction tuning on page 202](#).
- 3 Compensate for the friction using the instruction `FricIdSetFricLevels`.

3 Motion Performance

3.4.2 Automatic friction tuning

3.4.2 Automatic friction tuning

About automatic friction tuning

A robot's joint friction levels are automatically tuned with the instructions `FricIdInit` and `FricIdEvaluate`. These instructions will tune each joint's friction level for a specific sequence of movements.

The automatically tuned levels are applied for friction compensation with the instruction `FricIdSetFricLevels`.

Program execution

To perform automatic tuning for a sequence of movements, the sequence must begin with the instruction `FricIdInit` and end with the instruction `FricIdEvaluate`. When program execution reaches `FricIdEvaluate`, the robot will repeat the movement sequence until the best friction level for each joint axis is found. Each iteration consists of a backward and a forward motion, both following the programmed path. Typically the sequence has to be repeated approximately 20-30 times, in order to iterate to correct joint friction levels.

If the program execution is stopped in any way while the program pointer is on the instruction `FricIdEvaluate` and then restarted, the results will be invalid. After a stop, friction identification must therefore be restarted from the beginning.

Once the correct friction levels are found they have to be set with the instruction `FricIdSetFricLevels`, otherwise they will not be used. Note that the friction levels are tuned for the particular movement between `FricIdInit` and `FricIdEvaluate`. For movements in another region in the robot's working area, a new tuning is needed to obtain the correct friction levels.

For a detailed description of the instructions, see *Technical reference manual - RAPID Instructions, Functions and Data types*.

Limitations

There are the following limitations for friction tuning:

- Friction tuning cannot be combined with synchronized movement. That is, `SyncMoveOn` is not allowed between `FricIdInit` and `FricIdEvaluate`.
- The movement sequence for which friction tuning is done must begin and end with a finepoint. If not, finepoints will automatically be inserted during the tuning process.
- Automatic friction tuning works only for TCP robots.
- Automatic joint friction tuning can only be done for one robot at a time.
- Tuning can be made to a maximum of 500%. If that is not enough, set a higher value for the parameter *Friction FFW Level*, see [Starting with an estimated value on page 207](#).
- It is not possible to view any test signals with TuneMaster during automatic friction tuning.
- The movement sequence between `FricIdInit` and `FricIdEvaluate` cannot be longer than 10 seconds.

Continues on next page

**Note**

To use Advanced Shape Tuning, the parameter *Friction FFW On* must be set to TRUE.

Example

This example shows how to program a cutting instruction that encapsulates the friction tuning. When the instruction is run the first time, without calculated friction parameters, the friction tuning is done. During the tuning process, the robot will repeatedly move back and forth along the programmed path. Approximately 25 iterations are needed.

At all subsequent runs the friction levels are set to the tuned values identified in the first run. By using the instruction `CutHole`, the friction can be tuned individually for each hole.

```

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

```

**Note**

A real program would include deactivating the cutting equipment before the tuning phase.

3 Motion Performance

3.4.3 Manual friction tuning

3.4.3 Manual friction tuning

Overview

It is possible to make a manual tuning of a robot's joint friction (instead of automatic friction tuning). The friction level for each joint can be tuned using the instruction `TuneServo`. How to do this is described in this section.

There is usually no need to make changes to the friction ramp.



Note

To use Advanced Shape Tuning, the parameter *Friction FFW On* must be set to `TRUE`.

Tune types

A tune type is used as an argument to the instruction `TuneServo`. For more information, see *tunetype* in *Technical reference manual - RAPID Instructions, Functions and Data types*.

There are two tune types that are used expressly for Advanced Shape Tuning:

Tune type	Description
TUNE_FRIC_LEV	By calling the instruction <code>TuneServo</code> with the argument <code>TUNE_FRIC_LEV</code> the friction level for a robot joint can be adjusted during program execution. A value is given in percent (between 1 and 500) of the friction level defined by the parameter <i>Friction FFW Level</i> .
TUNE_FRIC_RAMP	By calling the instruction <code>TuneServo</code> with the argument <code>TUNE_FRIC_RAMP</code> the motor shaft speed at which full friction compensation is reached can be adjusted during program execution. A value is given in percent (between 1 and 500) of the friction ramp defined by the parameter <i>Friction FFW Ramp</i> . There is normally no need to tune the friction ramp.

Configure friction level

The friction level is set for each robot joint. Perform the following steps for one joint at a time:

	Action
1	Test the robot by running it through the most demanding parts of its tasks (the most advanced shapes). If the robot shall be used for cutting, then test it by cutting with the same tool as at manufacturing. Observe the path deviations and test if the joint friction levels need to be increased or decreased.
2	Tune the friction level with the RAPID instruction <code>TuneServo</code> and the tune type <code>TUNE_FRIC_LEV</code> . The level is given in percent of the <i>Friction FFW Level</i> value. Example: The instruction for increasing the friction level with 20% looks like this: <code>TuneServo MHA160R1, 1, 120 \Type:= TUNE_FRIC_LEV;</code>
3	Repeat step 1 and 2 until you are satisfied with the path deviation.

Continues on next page

	Action
4	The final tuning values can be transferred to the system parameters. Example: The <i>Friction FFW Level</i> is 0.5 and the final tune value (<code>TUNE_FRIC_LEV</code>) is 120%. Set <i>Friction FFW Level</i> to 0.6 and tune value to 100% (default value), which is equivalent.



Tip

Tuning can be made to a maximum of 500%. If that is not enough, set a higher value for the parameter *Friction FFW Level*, see [Setting tuning system parameters on page 207](#).

3 Motion Performance

3.4.4.1 System parameters

3.4.4 System parameters

3.4.4.1 System parameters

About the system parameters

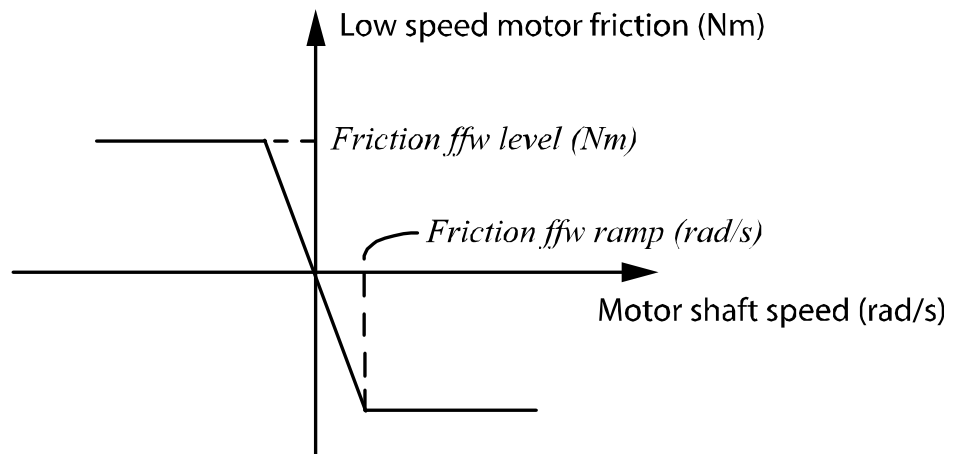
This is a brief description of each parameter in the option *Advanced Shape Tuning*. For more information, see the respective parameter in *Technical reference manual - System parameters*.

Friction Compensation / Control Parameters

These parameters belong to the type *Friction Compensation* in the topic *Motion*, except for the robots IRB 1400 and IRB 1410 where they belong to the type *Control Parameters* in the topic *Motion*.

Parameter	Description
Friction FFW On	Advanced Shape Tuning is active when <i>Friction FFW On</i> is set to TRUE.
Friction FFW Level	<i>Friction FFW Level</i> is the friction level for the robot joint. See illustration below.
Friction FFW Ramp	<i>Friction FFW Ramp</i> is the speed of the robot motor shaft, at which the friction has reached the friction level defined by <i>Friction FFW Level</i> . See illustration below. There is normally no need to make changes to <i>Friction FFW Ramp</i> .

Illustration



en0900000117

3.4.4.2 Setting tuning system parameters

Automatic tuning rarely requires changes in system parameters

For automatic tuning, if the friction levels are saved in a persistent array, the tuning is maintained after a power failure. The automatic tuning can also be used to set different tuning levels for different robot movement sequences, which cannot be achieved with system parameters. When using automatic tuning, there is no need to change the system parameters unless the default values are very much off, see [Starting with an estimated value on page 207](#).

Transfer tuning to system parameters

When using manual tuning, the tuning values are reset to default (100%) at power failure. System parameter settings are, however, permanent.

If a temporary tuning is made, that is only valid for a part of the program execution, it should not be transferred.

To transfer the friction level tuning value (`TUNE_FRIC_LEV`) to the parameter *Friction FFW Level* follow these steps:

	Action
1	In RobotStudio, open the Configuration Editor, Motion topic, and select the type Friction comp (except for the robots IRB 1400 and IRB 1410 where they belong to the type Control parameters).
2	Multiply <i>Friction FFW Level</i> with the tuning value. Set this value as the new <i>Friction FFW Level</i> and set the tuning value (<code>TUNE_FRIC_LEV</code>) to 100%. Example: The <i>Friction FFW Level</i> is 0.5 and the final tune value (<code>TUNE_FRIC_LEV</code>) is 120%. Set <i>Friction FFW Level</i> to 0.6 (1.20x0.5) and the tuning value to 100% (default value), which is equivalent.
3	Restart the controller for the changes to take effect.

Starting with an estimated value

The parameter *Friction FFW Level* will be the starting value for the tuning. If this value is very far from the correct value, tuning to the correct value might be impossible. This is unlikely to happen, since *Friction FFW Level* is by default set to a value approximately correct for most situations.

If the *Friction FFW Level* value, for some reason, is too far from the correct value, it can be changed to a new estimated value.

	Action
1	In RobotStudio, open the Configuration Editor, Motion topic, and select the type Friction comp (except for the robots IRB 1400 and IRB 1410 where they belong to the type Control parameters).
2	Set the parameter <i>Friction FFW Level</i> to an estimated value. Do not set the value 0 (zero), because that will make tuning impossible.
3	Restart the controller for the changes to take effect.

3 Motion Performance

3.4.5 RAPID components

3.4.5 RAPID components

About the RAPID components

This is an overview of all instructions, functions, and data types in *Advanced Shape Tuning*.

For more information, see *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instructions

Instructions	Description
FricIdInit	Initiate friction identification
FricIdEvaluate	Evaluate friction identification
FricIdSetFricLevels	Set friction levels after friction identification

Functions

Advanced Shape Tuning includes no functions.

Data types

Advanced Shape Tuning includes no data types.

3.5 Motion Process Mode [included in 3100-1]

3.5.1 About Motion Process Mode

Purpose

The purpose of Motion Process Mode is to simplify application specific tuning, i.e. to optimize the performance of the robot for a specific application.

For most applications the default mode is the best choice.



Tip

If the default mode does not give sufficient accuracy, first test to use *Accuracy mode*, and if that is not sufficient, use *Low speed accuracy*.

Available motion process modes

A motion process mode consists of a specific set of tuning parameters for a robot. Each tuning parameter set, that is each mode, optimizes the robot tuning for a specific class of applications.

There following modes are predefined:

- *Optimal cycle time mode* – this mode gives the shortest possible cycle time and is normally the default mode.
- *Accuracy mode* – this mode improves path accuracy. The cycle time will be slightly increased compared to *Optimal cycle time mode*.
- *Low speed accuracy mode* – this mode improves path accuracy. The cycle time will be slightly increased compared to *Accuracy mode*.
- *Low speed stiff mode* - this mode is recommended for contact applications where maximum servo stiffness is important. Could also be used in some low speed applications, where a minimum of path vibrations is desired. The cycle time will be increased compared to *Low speed accuracy mode*.
- *Press tending mode* – Changes the *Kv Factor*, *Kp Factor* and *Ti Factor* in order to mitigate tool vibrations. This mode is primarily intended for use in press tending applications where flexible grippers with a large extension in the y-direction are used.
- *Collaborative mode* – This mode is recommended for collaborative applications where robot should run smoothly. The cycle time will be increased compared to optimal cycle time mode. This will only have any effect on GoFa CRB 15000.

There are also four modes available for application specific user tuning:

- *MPM User mode 1 – 4*

Selection of mode

The default mode is automatically selected and can be changed by changing the system parameter *Use Motion Process Mode* for type *Robot*.

Continues on next page

3 Motion Performance

3.5.1 About Motion Process Mode

Continued

Changing the *Motion Process Mode* from RAPID is only possible if the option *Advanced Robot Motion* is installed. The mode can only be changed when the robot is standing still, otherwise a fine point is enforced.

The following example shows a typical use of the RAPID instruction

MotionProcessModeSet.

```
MotionProcessModeSet OPTIMAL_CYCLE_TIME_MODE;  
! Do cycle-time critical movement  
MoveL *, v $\mathbf{max}$ , ...;  
...
```

```
MotionProcessModeSet ACCURACY_MODE;  
! Do cutting with high accuracy  
MoveL *, v $\mathbf{50}$ , ...;  
...
```

Limitations

- The *Motion Process Mode* concept is currently available for all six- and seven-axes robots except paint robots with TrueMove1.
- The *Mounting Stiffness Factor* parameters are only available for the following robots:
IRB 120, IRB 140, IRB 1200, IRB 1520, IRB 1600, IRB 2600, IRB 4600, IRB 6620 (not LX), IRB 6640, IRB 6700.
- For IRB 1410, only the *Accset* and the geometric accuracy parameters are available.
- The following robot models do not support the use of *World Acc Factor* (i.e. only *World Acc Factor* = -1 is allowed):
IRB 340, IRB 360, IRB 540, IRB 1400, IRB 1410

3.5.2 User-defined modes

Available tune parameters

If a more specific tuning is needed, some tuning parameters can be modified in each motion process mode. The predefined modes and the user modes can all be modified. In this way, the user can create a specific tuning for a specific application. The following list contains a short description of the available tune parameters.

- *Use Motion Process Mode Type* - selects predefined parameters for a user mode.
- *Accset Acc Factor* – changes acceleration
- *Accset Ramp Factor* – changes acceleration ramp
- *Accset Fine Point Ramp Factor* – changes deceleration ramp in fine points
- *Joint Acc Factor* - changes acceleration for a specific joint.
- *World Acc Factor* - activates dynamic world acceleration limitation if positive, typical value is 1, deactivated if -1.
- *Geometric Accuracy Factor* - improves geometric accuracy if reduced.
- *Dh Factor* – changes path smoothness (effective system bandwidth)
- *Df Factor* – changes the predicted resonance frequency for a particular axis
- *Kp Factor* – changes the equivalent gain of the position controller for a particular axis
- *Kv Factor* – changes the equivalent gain of the speed controller for a particular axis
- *Ti Factor* – changes the integral time of the controller for a particular axis
- *Mounting Stiffness Factor X* – describes the stiffness of the robot foundation in x direction
- *Mounting Stiffness Factor Y* – describes the stiffness of the robot foundation in y direction
- *Mounting Stiffness Factor Z* – describes the stiffness of the robot foundation in z direction

For a detailed description, see *Motion Process Mode* in *Technical reference manual - System parameters*.

Tuning parameters from RAPID

Most parameters can also be changed using the `TuneServo` and `AccSet` instructions.



Note

All parameter settings are relative adjustments of the predefined parameter values. Although it is possible to combine the use of motion process modes and `TuneServo/Accset` instructions, it is recommended to choose either motion process modes or `TuneServo/AccSet`.

Continues on next page

3 Motion Performance

3.5.2 User-defined modes

Continued

Example 1

Relative adjustment of acceleration = $[Predefined\ AccSet\ Acc\ Factor] * [AccSet\ Acc\ Factor] * [AccSet\ instruction\ acceleration\ factor / 100]$

Example 2

Relative adjustment of Kv = $[Predefined\ Kv\ Factor] * [Kv\ Factor] * [Tune\ value\ of\ TuneServo(TYPE_KV)\ instruction / 100]$

Predefined parameter values

The predefined parameter values for each mode varies for different robot types. Generally, all predefined parameters are set to 1.0 for *Optimal cycle time mode*. For *Low speed accuracy mode* and *Low speed stiff mode*, the `AccSet` and `Dh` parameters are lowered for a smoother movement and a more accurate path, and the *Kv Factor*, *Kp Factor*, and *Ti Factor* are changed for higher servo stiffness.

For some robots, it might not be possible to increase the *Kv Factor* in *Low speed accuracy mode* and *Low speed stiff mode*. Always be careful and be observant for increased motor noise level when adjusting *Kv Factor* and do not use higher values than needed for fulfilling the application requirement. A *Kp Factor* which is too high, or a *Ti Factor* which is too low, can also increase vibrations due to mechanical resonances.

Accuracy Mode uses a dynamic world acceleration limitation (*World Acc Factor*) and increased geometric accuracy (*Geometric Accuracy Factor*) to improve the path accuracy.

The *Df Factor* and the *Mounting Stiffness Factors* are always set to 1.0 in the predefined modes, since the optimal values of these parameters depends the specific installation, for example, the stiffness of the foundation on which the robot is mounted. These parameters can be optimized using *TuneMaster*. More information can be found in the *TuneMaster* application. Also note the limitations of *Mounting Stiffness Factor*.



WARNING

Incorrect setting of the *Motion Process Mode* parameters can cause oscillating movements or torques that can damage the robot.

3.5.3 General information about robot tuning

Minimizing cycle time

For best possible cycle time, the motion process mode *Optimal cycle time mode* should be used. This mode is normally the default mode. The user only needs to define the tool load, payload, and arm loads if any. Once the robot path has been programmed, the *ABB QuickMove* motion technology automatically computes the optimal accelerations and speeds along the path. This results in a time-optimal path with the shortest possible cycle time. Hence, no tuning of acceleration is needed. The only way to improve the cycle time is to change the geometry of the path or to work in another region of the work space. This type of optimization, if needed, can be performed by simulation in RobotStudio.

Increasing path accuracy and reducing vibrations

For most applications, the *Optimal cycle time mode* will result in a satisfactory behavior in terms of path accuracy and vibrations. This is due to the *ABB TrueMove* motion technology. However, there are applications where the accuracy needs to be improved by modifying the tuning of the robot. This tuning has previously been performed by using the `TuneServo` and `AccSet` instructions in the RAPID program. The concept of motion process modes will simplify this application specific tuning and the four predefined modes should be useful in many cases with no further adjustments needed.

Here follows some general advice for solving accuracy problems, assuming that the default choice *Optimal cycle time mode* has been tested and that accuracy problems have been noticed:

- 1 Verify that tool load, payload, and arm loads are properly defined.
- 2 Inspect tool and process equipment attached to the robot arms. Make sure that everything is properly fastened and that rigidity of the tool is adequate.
- 3 Inspect the foundation on which the robot is mounted, see [Compensating for foundation flexibility on page 213](#).

Compensating for foundation flexibility

If the foundation does not fulfill the stiffness requirement of the robot product manual, then the foundation flexibility should be compensated for. See section *Requirements on foundation, Minimum resonance frequency* in the robot product manual.

This is performed by *Df Factor* for axis 1 and 2 or *Mounting Stiffness Factor* depending on robot type, see [Limitations on page 216](#).

Continues on next page

3 Motion Performance

3.5.3 General information about robot tuning

Continued

TuneMaster is used for finding the optimal value of *Df Factor / Mounting Stiffness Factor*. The obtained *Df Factor / Mounting Stiffness Factor* is then defined for the *Motion Process Modes* used.



Note

A foundation that does not fulfill the requirements always impairs the accuracy to some extent, even if the described compensation is used. If the foundation rigidity is very low, there might not be possible to solve the problem using *Df Factor / Mounting Stiffness Factor*.

In this case, the foundation must be improved or any of the solutions below used, for example, *Optimal cycle time mode* with a low *Dh Factor*, *Accset Acc Factor*, or *Accset Fine Point Ramp Factor* depending on the application.



WARNING

Incorrect tuning for a very low mounting stiffness can cause oscillating movements or torques that can damage the robot.

If accuracy still needs to be improved

- For applications with high demands on path accuracy, for example cutting, *Advanced Shape Tuning* and *Accuracy mode/Low speed accuracy mode* should be used. The choice of motion mode depends both on the robot type and the specific application. In general, *Accuracy mode* is recommended for small and medium size robots (up to *IRB 2400/2600*) and *Low speed accuracy mode* is recommended for larger robots.
- If the path accuracy still needs improvement, the accuracy modes can be adjusted with the tune parameters, some examples:
 - Tuning of *Accuracy mode* for improved accuracy:
 - 1) Reduce *World Acc Factor*, for example from 1 to 0.5.
 - 2) Reduce *Dh Factor* to 0.5 or lower. Note that a low value of *Dh factor* can change the corner zones at high speed.
 - Tuning of *Low speed accuracy mode* for improved accuracy:
 - 1) Set *World Acc Factor* to 1, and set *Geometric Accuracy Factor* to 0.1.
 - 2) Reduce *Dh Factor* to 0.5 or lower.
- The programmed speed must sometimes be reduced for best possible accuracy, e.g. in cutting applications. For example, a circle with radius 1 mm should not be programmed with a higher speed than 20 mm/s.
- For contact applications, for example milling and pre-machining, *Low speed stiff mode* is recommended. This mode can also be useful for large robots in some low speed applications (up to 100 mm/s) where a minimum of path vibrations is required, for example below 0.1 mm. Note that this mode has a very stiff servo tuning and that there may be cases where the *Kv Factor* needs to be reduced due to motor vibrations and noise.

Continues on next page

- If overshoots and vibrations in fine points needs to be reduced. Use *Optimal cycle time mode* and decrease the value of *Accset Fine Point Ramp Factor* or *Dh Factor* until the problem is solved.
- If accuracy problems occur when starting or ending reorientation. Define a new zone with increased `pzone_ori` and `pzone_eax`. These should always have the same value, even if there are no external axes in the system. Also increase `zone_ori`. Always strive for smooth reorientations when programming.
- Finally, if the cycle time needs to be reduced after the tuning for accuracy is finished. Use different motion process modes in different sections of the RAPID program.

3 Motion Performance

3.5.4 Additional information

3.5.4 Additional information

Motion Process Mode compared to TuneServo and AccSet

Motion process modes simplifies application specific tuning and makes it possible to define the tuning by system parameters instead of the RAPID program.

In general, motion process modes should be the first choice for solving accuracy problems. However, application specific tuning can still be performed using the `TuneServo` and `AccSet` instructions in the RAPID program.

There are a few situations where `TuneServo` and `AccSet` might be a better choice. One example of this is if an acceleration reduction in a section of the RAPID program solves the accuracy problem and the cycle time is to be optimized. In this case it might be better to use `AccSet` which can be changed without fine point whereas change of motion process mode requires a fine point.

Limitations

- The *Motion Process Mode* concept is currently available for all six- and seven-axes robots except paint robots.
- The *Mounting Stiffness Factor* parameters are only available for the following robots:
IRB 120, IRB 140, IRB 1200, IRB 1520, IRB 1600, IRB 2600, IRB 4600, IRB 6620 (not LX), IRB 6640, IRB 6700.
- For IRB 1410, only the *Accset* and the geometric accuracy parameters are available.
- The following robot models do not support the use of *World Acc Factor* (i.e. only *World Acc Factor = -1* is allowed):
IRB 340, IRB 360, IRB 540, IRB 1400, IRB 1410

Related information

For information about	See
Configuration of <i>Motion Process Mode</i> parameters.	<i>Technical reference manual - System parameters</i>
RAPID instructions: <ul style="list-style-type: none">• <code>AccSet</code> - Reduces the acceleration• <code>MotionProcessModeSet</code> - Set motion process mode• <code>TuneServo</code> - Tuning servos	<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>

3.6 Wrist Move [included in 3100-1]

3.6.1 Introduction to Wrist Move

Purpose

The purpose of *Wrist Move* is to improve the path accuracy when cutting geometries with small dimensions. For geometrical shapes like small holes, friction effects from the main axes (1-3) of the robot often degrade the visual appearance of the shape. The key idea is that instead of controlling the robot's TCP, a wrist movement controls the point of intersection between the laser beam (or water jet or routing spindle, etc) and the cutting plane. For controlling the point of intersection, only two wrist axes are needed. Instead of using all axes of the robot, only two wrist axes are used, thereby minimizing the friction effects on the path. Which wrist axis pair to be used is decided by the programmer.

Using Wrist Move

Wrist Move is included in the RobotWare option *Advanced robot motion*.

Wrist Move is used together with the RAPID instruction `CirPathMode` and movement instructions for circular arcs, that is, `MoveC`, `TrigC`, `CapC` etc. The wrist movement mode is activated by the instruction `CirPathMode` together with one of the flags `Wrist45`, `Wrist46`, or `Wrist56`. With this mode activated, all subsequent `MoveC` instructions will result in a wrist movement. To go back to normal `MoveC` behavior, then `CirPathMode` has to be set with a flag other than `Wrist45`, `Wrist46`, and `Wrist56`, for example, `PathFrame`.



Note

During a wrist movement, the TCP height above the surface will vary. This is an unavoidable consequence of using only two axes. The height variation will depend on the robot position, the tool definition, and the radius of the circular arc. The larger the radius, the larger the height variation will be. Due to the height variation it is recommended that the movement is run at a very low speed the first time to verify that the height variation does not become too large. Otherwise it is possible that the cutting tool collides with the surface being cut.

Limitations

The *Wrist Move* option cannot be used if:

- The work object is moving
- The robot is mounted on a track or another manipulator that is moving

The *Wrist Move* option is only supported for robots running *QuickMove*, second generation.

The tool will not remain at right angle against the surface during the cutting. As a consequence, the holes cut with this method will be slightly conical. Usually this will not be a problem for thin plates, but for thick plates the conicity will become apparent.

Continues on next page

3 Motion Performance

3.6.1 Introduction to Wrist Move

Continued

The height of the TCP above the surface will vary during the cut. The height variation will increase with the size of the shape being cut. What limits the possible size of the shape are therefore, beside risk of collision, process characteristics like focal length of the laser beam or the water jet.

WristMove cannot be used on robots with non-spherical wrist, for example, GoFa or YuMi

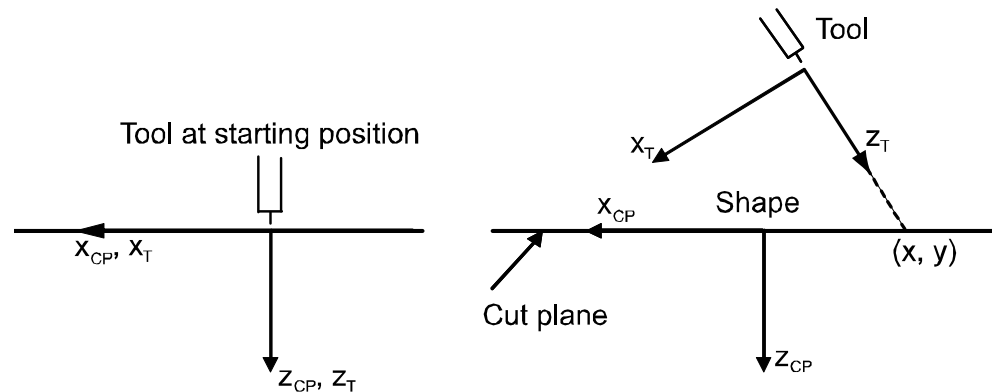
3.6.2 Cut plane frame

Defining the cut plane frame

Crucial to the wrist movement concept is the definition of the cut plane frame. This frame provides information about position and orientation of the object surface. The cut plane frame is defined by the robot's starting position when executing a `MoveC` instruction. The frame is defined to be equal to the tool frame at the starting position. Note that for a sequence of `MoveC` instructions, the cut plane frame stays the same during the whole sequence.

Illustration, cut plane

The left illustration shows how the cut plane is defined, and the right illustration shows the tool- and cut plane frames during cutting.



en0900000118

Prerequisites

Due to the way the cut plane frame is defined, the following must be fulfilled at the starting position:

- The tool must be at right angle to the surface
- The z-axis of the tool must coincide with the laser beam or water jet
- The TCP must be as close to the surface as possible

If the first two requirements are not fulfilled, then the shape of the cut contour will be affected. For example, a circular hole would look more like an ellipse. The third requirement is normally easy to fulfill as the TCP is often defined to be a few mm in front of, for example, the nozzle of a water jet. However, if the third requirement is not fulfilled, then it will only affect the radius of the resulting circle arc. That is, the radius of the cut arc will not agree with the programmed radius. For a linear segment, the length will be affected.



Tip

In the jog window of the FlexPendant there is a button for automatic alignment of the tool against a chosen coordinate frame. This functionality can be used to ensure that the tool is at a right angle against the surface when starting the wrist movement.

Continues on next page

3 Motion Performance

3.6.2 Cut plane frame

Continued



Tip

Wrist movement is not limited to circular arcs only: If the targets of MoveC are collinear, then a straight line will be achieved.

3.6.3 RAPID components

Instruction

This is a brief description of the instruction used in Wrist Move. For more information, see the description of the instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Descriptions
CirPathMode	<p>CirPathMode makes it possible to select different modes to reorientate the tool during circular movements.</p> <p>The arguments <code>Wrist45</code>, <code>Wrist46</code>, and <code>Wrist56</code> are used specifically for the Wrist Move option.</p>

3 Motion Performance

3.6.4 RAPID code, examples

3.6.4 RAPID code, examples

Basic example

This example shows how to do two circular arcs, first using axes 4 and 5, and then using axes 5 and 6. After the two arcs, wrist movement is deactivated by `CirPathMode`.

```
! 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;
```

Advanced example

This example shows how to cut a slot with end radius R and length $L+2R$, using wrist movement. See [Illustration, pSlot and wSlot on page 223](#). The slot both begins and ends at the position `pSlot`, which is the center of the left semi-circle. To avoid introducing oscillations in the robot, the cut begins and ends with semi-circular lead-in and lead-out paths that connect smoothly to the slot contour. All coordinates are given relative the work object `wSlot`.

```
! 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;
```

Continues on next page

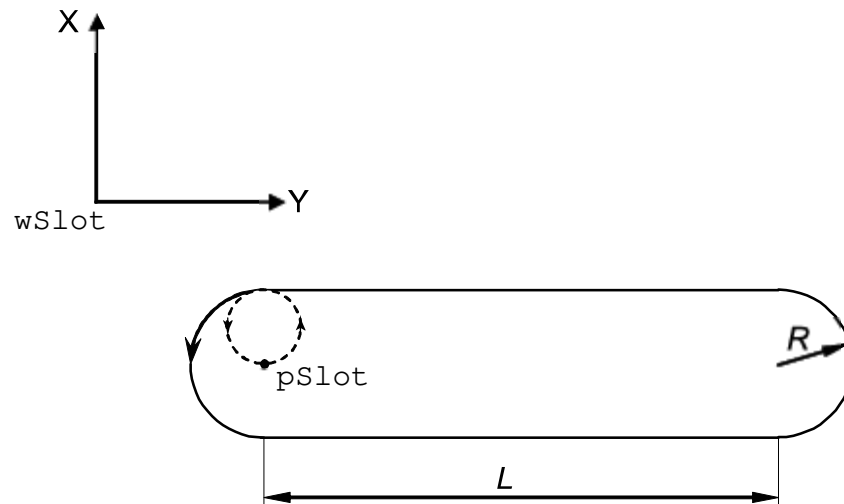
```
! Right semi-circle
MoveC Offs(pSlot, L+R, 0, 0), Offs(pSlot, L, R, 0), v50, z0, tLaser,
      \wobj := wSlot;

! 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;
```

Illustration, pSlot and wSlot



xx0900000111

3 Motion Performance

3.6.5 Troubleshooting

3.6.5 Troubleshooting

Unexpected cut shape

If the cut shape is not the expected, then check the following:

- The tool z-axis coincides with the laser beam or the water jet
- The tool z-axis is at right angle to the surface at the starting position of the first `MoveC`
- If you have the option Advanced Shape Tuning, then try tuning the friction for the involved wrist axes.

Mismatching radius

If the radius of the circular arc does not agree with the programmed radius, then check that the TCP is as close to the surface as possible at the starting position.

Impossible movement with chosen axis pair

If the movement is not possible with the selected axis pair, then try activating another pair by using one of the flags `Wrist45`, `Wrist46`, or `Wrist56`. As a last resort, try reaching the starting position with another robot configuration.

4 Motion Supervision

4.1 World Zones [3106-1]

4.1.1 Overview of World Zones

Purpose

The purpose of World Zones is to stop the robot or set an output signal if the robot is inside a special user-defined zone. Here are some examples of applications:

- When two robots share a part of their respective work areas. The possibility of the two robots colliding can be safely eliminated by World Zones supervision.
- When a permanent obstacle or some temporary external equipment is located inside the robot's work area. A forbidden zone can be created to prevent the robot from colliding with this equipment.
- Indication that the robot is at a position where it is permissible to start program execution from a Programmable Logic Controller (PLC).

A world zone is supervised during robot movements both during program execution and jogging. If the robot's TCP reaches the world zone or if the axes reaches the world zone in joints, the movement is stopped or a digital output signal is set.



WARNING

For safety reasons, this software shall not be used for protection of personnel. Use hardware protection equipment for that.

What is included

The RobotWare option World Zones gives you access to:

- instructions used to define volumes of various shapes
- instructions used to define joint zones in coordinates for axes
- instructions used to define and enable world zones

Basic approach

This is the general approach for setting up World Zones. For a more detailed example of how this is done, see [Code examples on page 229](#).

- 1 Declare the world zone as stationary or temporary.
- 2 Declare the shape variable.
- 3 Define the shape that the world zone shall have.
- 4 Define the world zone (that the robot shall stop or that an output signal shall be set when reaching the volume).

Continues on next page

4 Motion Supervision

4.1.1 Overview of World Zones

Continued

Limitations

Supervision of a volume only works for the TCP. Any other part of the robot may pass through the volume undetected. To be certain to prevent this, you can supervise a joint world zone (defined by `WZLimJointDef` or `WZHomeJointDef`).

A variable of type `wzstationary` or `wztemporary` can not be redefined. They can only be defined once (with `WZLimSup` or `WZDOSet`).

World Zones supervision is not accessible when lead-through is active.

4.1.2 RAPID components

Data types

This is a brief description of each data type in World Zones. For more information, see respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
wztemporary	<p>wztemporary is used to identify a temporary world zone and can be used anywhere in the RAPID program.</p> <p>Temporary world zones can be disabled, enabled again, or erased via RAPID instructions. Temporary world zones are automatically erased when a new program is loaded or when program execution start from the beginning in the MAIN routine.</p>
wzstationary	<p>wzstationary is used to identify a stationary world zone and can only be used in an event routine connected to the event POWER ON. For information on defining event routines, see <i>Operating manual - OmniCore</i>.</p> <p>A stationary world zone is always active and is reactivated by a restart (switch power off then on, or change system parameters). It is not possible to disable, enable or erase a stationary world zone via RAPID instructions.</p> <p>Stationary world zones shall be used if security is involved.</p>
shapedata	<p>shapedata is used to describe the geometry of a world zone.</p> <p>World zones can be defined in 4 different geometrical shapes:</p> <ul style="list-style-type: none"> • a straight box, with all sides parallel to the world coordinate system • a cylinder, parallel to the z axis of the world coordinate system • a sphere • a joint angle area for the robot axes and/or external axes

Instructions

This is a brief description of each instruction in World Zones. For more information, see respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
WZBoxDef	<p>WZBoxDef is used to define a volume that has the shape of a straight box with all its sides parallel to the axes of the world coordinate system. The definition is stored in a variable of type <code>shapedata</code>.</p> <p>The volume can also be defined as the inverse of the box (all volume outside the box).</p>
WZCylDef	<p>WZCylDef is used to define a volume that has the shape of a cylinder with the cylinder axis parallel to the z-axis of the world coordinate system. The definition is stored in a variable of type <code>shapedata</code>.</p> <p>The volume can also be defined as the inverse of the cylinder (all volume outside the cylinder).</p>
WZSphDef	<p>WZSphDef is used to define a volume that has the shape of a sphere. The definition is stored in a variable of type <code>shapedata</code>.</p> <p>The volume can also be defined as the inverse of the sphere (all volume outside the sphere).</p>

Continues on next page

4 Motion Supervision

4.1.2 RAPID components

Continued

Instruction	Description
WZLimJointDef	<p>WZLimJointDef is used to define joint coordinate for axes, to be used for limitation of the working area. Coordinate limits can be set for both the robot axes and external axes.</p> <p>For each axis WZLimJointDef defines an upper and lower limit. For rotational axes the limits are given in degrees and for linear axes the limits are given in mm.</p> <p>The definition is stored in a variable of type <code>shapedata</code>.</p>
WZHomeJointDef	<p>WZHomeJointDef is used to define joint coordinates for axes, to be used to identify a position in the joint space. Coordinate limits can be set for both the robot axes and external axes.</p> <p>For each axis WZHomeJointDef defines a joint coordinate for the middle of the zone and the zones delta deviation from the middle. For rotational axes the coordinates are given in degrees and for linear axes the coordinates are given in mm.</p> <p>The definition is stored in a variable of type <code>shapedata</code>.</p>
WZLimSup	<p>WZLimSup is used to define, and enable, stopping the robot with an error message when the TCP reaches the world zone. This supervision is active both during program execution and when jogging.</p> <p>When calling WZLimSup you specify whether it is a stationary world zone, stored in a <code>wzstationary</code> variable, or a temporary world zone, stored in a <code>wztemporary</code> variable.</p>
WZDOSet	<p>WZDOSet is used to define, and enable, setting a digital output signal when the TCP reaches the world zone.</p> <p>When calling WZDOSet you specify whether it is a stationary world zone, stored in a <code>wzstationary</code> variable, or a temporary world zone, stored in a <code>wztemporary</code> variable.</p>
WZDisable	<p>WZDisable is used to disable the supervision of a temporary world zone.</p>
WZEnable	<p>WZEnable is used to re-enable the supervision of a temporary world zone.</p> <p>A world zone is automatically enabled on creation. Enabling is only necessary after it has been disabled with WZDisable.</p>
WZFree	<p>WZFree is used to disable and erase a temporary world zone.</p>

Functions

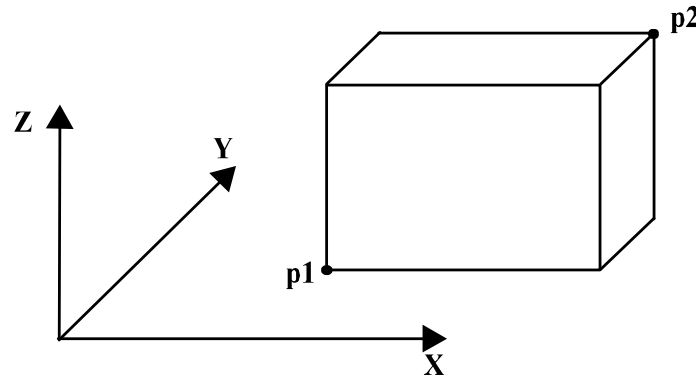
World Zones does not include any RAPID functions.

4.1.3 Code examples

Create protected box

To prevent the robot TCP from moving into stationary equipment, set up a stationary world zone around the equipment.

The routine `my_power_on` should then be connected to the event `POWER ON`. For information on how to do this, read about defining event routines in *Operating manual - OmniCore*.



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

```

Signal when robot is in position

When two robots share a work area it is important to know when a robot is out of the way, letting the other robot move freely.

This example defines a home position where the robot is in a safe position and sets an output signal when the robot is in its home position. The robot is standing on a travel track, handled as external axis 1. No other external axes are active.

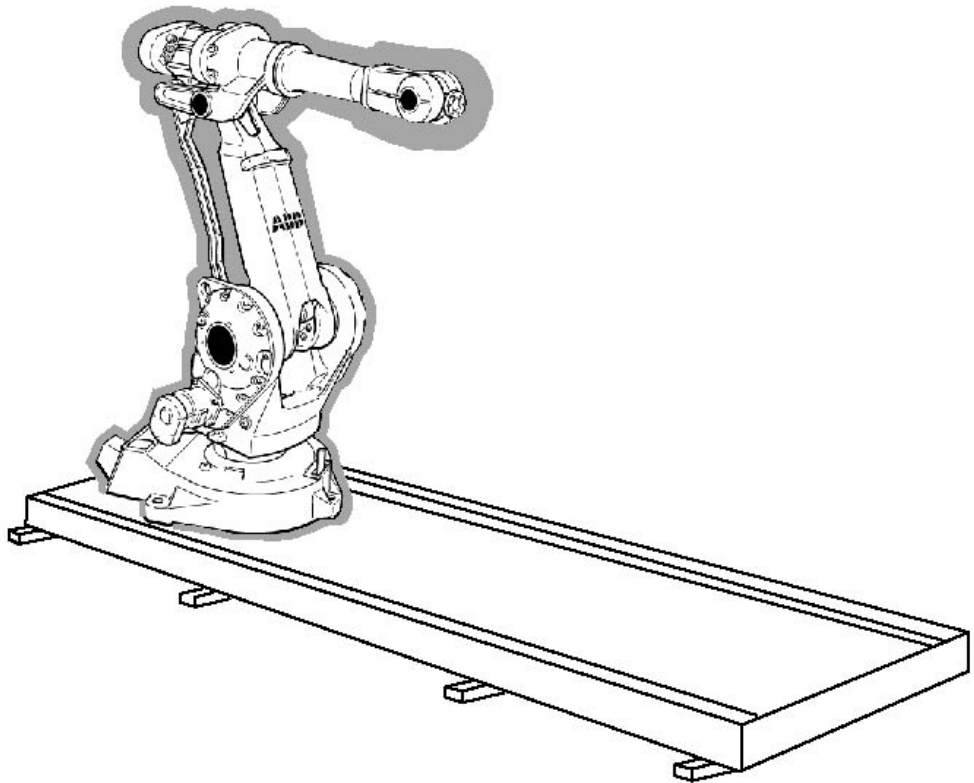
Continues on next page

4 Motion Supervision

4.1.3 Code examples

Continued

The shadowed area in the illustration shows the world zone.



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
```

4.2 Collision Detection [3107-1]

4.2.1 Overview

Purpose

Collision Detection is a software option that reduces collision impact forces on the robot. This helps protecting the robot and external equipment from severe damage.



WARNING

Collision Detection cannot protect equipment from damage at a full speed collision.

Description

The software option Collision Detection identifies a collision by high sensitivity, model based supervision of the robot. Depending on what forces you deliberately apply on the robot, the sensitivity can be tuned as well as turned on and off. Because the forces on the robot can vary during program execution, the sensitivity can be set on-line in the program code.

Collision detection is more sensitive than the ordinary supervision and has extra features. When a collision is detected, the robot will immediately stop and relieve the residual forces by moving in reversed direction a short distance along its path. After a collision error message has been acknowledged, the movement can continue without having to press **Motors on** on the controller.

What is included

The RobotWare option Collision Detection gives you access to:

- system parameters for defining if Collision Detection should be active and how sensitive it should be (without the option you can only turn detection on and off for Auto mode)
- instruction for on-line changes of the sensitivity: `MotionSup`

Basic approach

Collision Detection is by default always active when the robot is moving. In many cases this means that you can use Collision Detection without having to take any active measures.

If necessary, you can turn Collision Detection on and off or change its sensitivity in two ways:

- temporary changes can be made on-line with the RAPID instruction `MotionSup`
- permanent changes are made through the system parameters.

Continues on next page

4 Motion Supervision

4.2.1 Overview

Continued

Collision detection for YuMi robots

As default YuMi will have collision detection active at stand still. It also has another stop ramp compared to other robots to be able to release clamping forces.



Note

If the tool data is wrong, false collisions might be triggered and the robot arm might drop a short distance during the stop ramp.

4.2.2 Limitations

Load definition

In order to detect collisions properly, the payload of the robot must be correctly defined.

**Tip**

Use Load Identification to define the payload. For more information, see *Operating manual - OmniCore*.

Robot axes only

Collision Detection is only available for the robot axes. It is not available for track motions, orbit stations, or any other external axes.

Independent joint

The collision detection is deactivated when at least one axis is run in independent joint mode. This is also the case even when it is an external axis that is run as an independent joint.

Soft servo

The collision detection may trigger without a collision when the robot is used in soft servo mode. Therefore, it is recommended to turn the collision detection off when the robot is in soft servo mode.

No change until the robot moves

If the RAPID instruction `MotionSup` is used to turn off the collision detection, this will only take effect once the robot starts to move. As a result, the digital output `MotSupOn` may temporarily have an unexpected value at program start before the robot starts to move.

Reversed movement distance

The distance the robot is reversed after a collision is proportional to the speed of the motion before the collision. If repeated low speed collisions occur, the robot may not be reversed sufficiently to relieve the stress of the collision. As a result, it may not be possible to jog the robot without the supervision triggering. In this case, turn Collision Detection off temporarily and jog the robot away from the obstacle.

Delay before reversed movement

In the event of a stiff collision during program execution, it may take a few seconds before the robot starts the reversed movement.

Robot on track motion

If the robot is mounted on a track motion the collision detection should be deactivated when the track motion is moving. If it is not deactivated, the collision detection may trigger when the track moves, even if there is no collision.

4 Motion Supervision

4.2.3 What happens at a collision

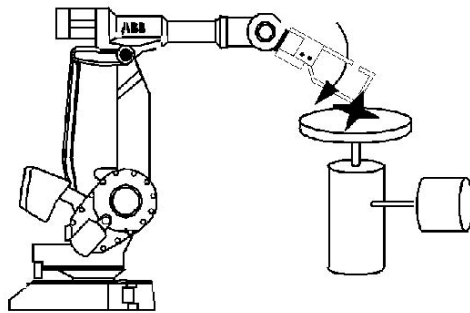
4.2.3 What happens at a collision

Overview

When the collision detection is triggered, the robot will stop as quickly as possible. Then it will move in the reverse direction to remove residual forces. The program execution will stop with an error message. The robot remains in the state *motors on* so that program execution can be resumed after the collision error message has been acknowledged.

A typical collision is illustrated below.

Collision illustration



xx0300000361

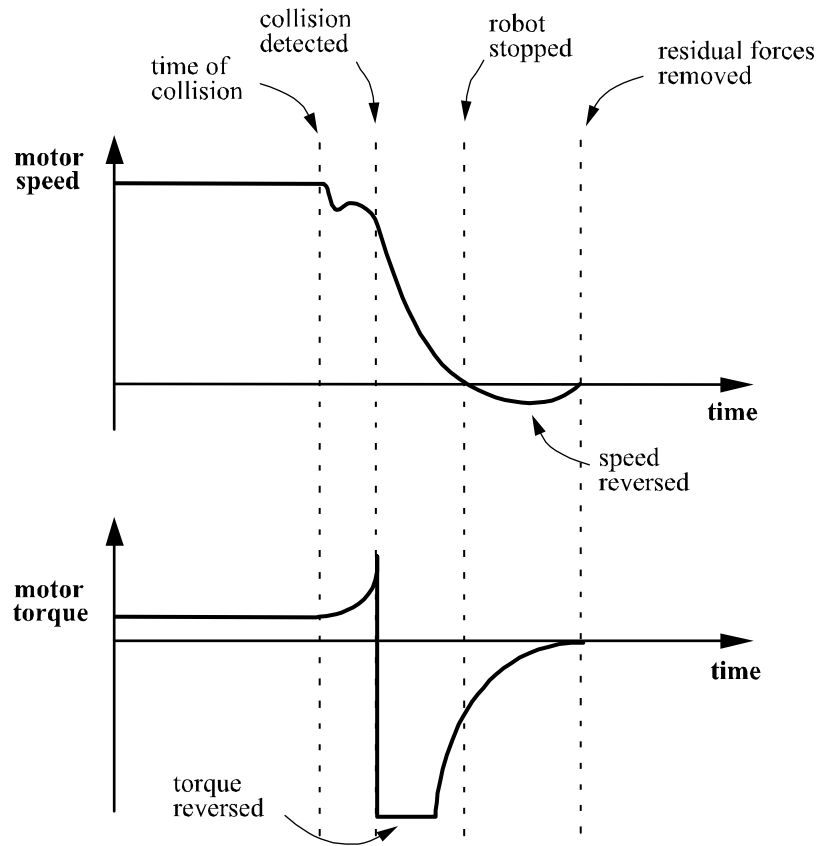
Robot behavior after a collision

This list shows the order of events after a collision. For an illustration of the sequence, see the diagram below.

When ...	then ...
the collision is detected	the motor torques are reversed and the mechanical brakes applied in order to stop the robot
the robot has stopped	the robot moves in reversed direction a short distance along the path in order to remove any residual forces which may be present if a collision or jam occurred
the residual forces are removed	the robot stops again and remains in the <i>motors on</i> state

Continues on next page

Speed and torque diagram



en0300000360

4 Motion Supervision

4.2.4 Additional information

4.2.4 Additional information

Motion error handling

For more information regarding error handling for a collision, see *Technical reference manual - RAPID kernel*.

4.2.5 Configuration and programming facilities

4.2.5.1 System parameters

About system parameters

Most of the system parameters for Collision Detection do **not** require a restart to take effect.

For more information about the parameters, see *Technical reference manual - System parameters*.

Motion Supervision

These parameters belong to the type *Motion Supervision* in the topic *Motion*.

Parameter	Description
Path Collision Detection	Turn the collision detection On or Off for program execution. <i>Path Collision Detection</i> is by default set to On.
Jog Collision Detection	Turn the collision detection On or Off for jogging. <i>Jog Collision Detection</i> is by default set to On.
Path Collision Detection Level	Modifies the Collision Detection supervision level for program execution by the specified percentage value. A large percentage value makes the function less sensitive. <i>Path Collision Detection Level</i> is by default set to 100%.
Jog Collision Detection Level	Modifies the Collision Detection supervision level for jogging by the specified percentage value. A large percentage value makes the function less sensitive. <i>Jog Collision Detection Level</i> is by default set to 100%.
Collision Detection Memory	Defines how much the robot moves in reversed direction on the path after a collision, specified in seconds. If the robot moved fast before the collision it will move away a larger distance than if the speed was slow. <i>Collision Detection Memory</i> is by default set to 75 ms.
Manipulator Supervision	Turns the supervision for the loose arm detection on or off for IRB 340 and IRB 360. A loose arm will stop the robot and cause an error message. <i>Manipulator Supervision</i> is by default set to On.
Manipulator Supervision Level	Modifies the supervision level for the loose arm detection for the manipulators IRB 340 and IRB 360. A large value makes the function less sensitive. <i>Manipulator Supervision Level</i> is by default value set to 100%.

Motion Planner

These parameters belong to the type *Motion Planner* in the topic *Motion*.

Parameter	Description
Motion Supervision Max Level	Set the maximum level to which the total collision detection tune level can be changed. It is by default set to 300%.

Continues on next page

4 Motion Supervision

4.2.5.1 System parameters

Continued

Motion System

This parameter belongs to the type *Motion System* in the topic *Motion*.

Parameter	Description
Ind collision stop without brake	This parameter is only valid for systems using the MultiMove option. If this parameter is set to TRUE, detected collisions will be handled independently in RAPID tasks that are executed independently. A restart is required for this parameter to take effect.

General RAPID

These parameters belong to the type *General RAPID* in the topic *Controller*.

Parameter	Description
Collision Error Handler	Enables RAPID error handling for collision. <i>Collision Error Handler</i> is default set to Off. For more information regarding error handling for a collision, see <i>Technical reference manual - RAPID kernel</i>

4.2.5.2 RAPID components

Instructions

This is a brief description of the instructions in Collision Detection. For more information, see respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
MotionSup	<p>MotionSup is used to:</p> <ul style="list-style-type: none">• activate or deactivate Collision Detection. This can only be done if the parameter <i>Path Collision Detection</i> is set to On.• modify the supervision level with a specified percentage value (1-300%). A large percentage value makes the function less sensitive.

4 Motion Supervision

4.2.5.3 Signals

4.2.5.3 Signals

Digital outputs

This is a brief description of the digital outputs in Collision Detection. For more information, see respective digital output in *Technical reference manual - System parameters*.

Digital output	Description
MotSupOn	<p><i>MotSupOn</i> is high when Collision Detection is active and low when it is not active.</p> <p>Note that a change in the state takes effect when a motion starts. Thus, if Collision Detection is active and the robot is moving, <i>MotSupOn</i> is high. If the robot is stopped and Collision Detection is turned off, <i>MotSupOn</i> is still high. When the robot starts to move, <i>MotSupOn</i> switches to low.</p> <p>Before the first Motors On order after a restart of the robot controller, <i>MotSupOn</i> will reflect the value of the corresponding system parameter <i>Path Collision Detection</i>:</p> <ul style="list-style-type: none">• If <i>Path Collision Detection</i> is set to <i>On</i>, <i>MotSupOn</i> will be high.• If <i>Path Collision Detection</i> is set to <i>Off</i>, <i>MotSupOn</i> will be low.
MotSupTrigg	<p><i>MotSupTrigg</i> goes high when the collision detection triggers. It stays high until the error code is acknowledged from the FlexPendant.</p>

4.2.6 How to use Collision Detection

4.2.6.1 Set up system parameters

Activate supervision

To be able to use Collision Detection during program execution, the parameter *Path Collision Detection* must be set to *On*.

To be able to use Collision Detection during jogging, the parameter *Jog Collision Detection* must be set to *On*.

Define supervision levels

Set the parameter *Path Collision Detection Level* to the percentage value you want as default during program execution.

Set the parameter *Jog Collision Detection Level* to the percentage value you want as default during jogging.

4 Motion Supervision

4.2.6.2 Adjust supervision from FlexPendant

4.2.6.2 Adjust supervision from FlexPendant

Speed adjusted supervision level

Collision Detection uses a variable supervision level. At low speeds it is more sensitive than at high speeds. For this reason, no tuning of the function should be required by the user during normal operating conditions. However, it is possible to turn the function on and off and to tune the supervision levels.

Separate tuning parameters are available for jogging and program execution. These parameters are described in [System parameters on page 237](#).

Set jog supervision on FlexPendant

On the FlexPendant, select **Control** from the **QuickSet** window and then tap **Jog**. On the **Jog Settings**, tap **Jog Supervision**.

Supervision can be turned on or off and the sensitivity can be adjusted for both programmed paths and jogging. The sensitivity level is set in percentage. A large value makes the function less sensitive.

If the motion supervision for jogging is turned off in the dialog box and a program is executed, Collision Detection can still be active during execution of the program.



Note

The supervision settings correspond to system parameters of the type *Motion Supervision*. These can be set using the supervision settings on the FlexPendant, as described above. They can also be changed using RobotStudio or FlexPendant configuration editor or Quickset Mechanical unit menu.

4.2.6.3 Adjust supervision from RAPID program

Default values

If Collision Detection is activated with the system parameters, it is by default active during program execution with the tune value 100%. These values are set automatically:

- when using the restart mode **Reset system**.
- when a new program is loaded.
- when starting program execution from the beginning.



Note

If tune values are set in the system parameters and in the RAPID instruction, both values are taken into consideration.

Example: If the tune value in the system parameters is set to 150% and the tune value is set to 200% in the RAPID instruction the resulting tune level will be 300%.

Temporarily deactivate supervision

If external forces will affect the robot during a part of the program execution, temporarily deactivate the supervision with the following instruction:

```
MotionSup \Off;
```

Reactivate supervision

If the supervision has been temporarily deactivated, it can be activated with the following instruction:

```
MotionSup \On;
```



Note

If the supervision is deactivated with the system parameters, it cannot be activated with RAPID instructions.

Tuning

The supervision level can be tuned during program execution with the instruction *MotionSup*. The tune values are set in percent of the basic tuning where 100% corresponds to the basic values. A higher percentage gives a less sensitive system.

This is an example of an instruction that increase the supervision level to 200%:

```
MotionSup \On \TuneValue:=200;
```

4 Motion Supervision

4.2.6.4 How to avoid false triggering

4.2.6.4 How to avoid false triggering

About false triggering

Because the supervision is designed to be very sensitive, it may trigger if the load data is incorrect or if there are large process forces acting on the robot.

Actions to take

If ...	then ...
the payload is incorrectly defined	use Load Identification to define it. For more information, see <i>Operating manual - OmniCore</i> .
the payload has large mass or inertia	increase supervision level
the arm load (cables or similar) cause trigger	manually define the arm load or increase supervision level
the application involves many external process forces	increase the supervision level for jogging and program execution in steps of 30 percent until you no longer receive the error code.
the external process forces are only temporary	use the instruction <code>MotionSup</code> to raise the supervision level or turn the function off temporarily.
everything else fails	turn off Collision Detection.

4.3 Collision Avoidance [3150-1]

Introduction

The function *Collision Avoidance* monitors a detailed geometric model of the robot. By defining additional geometrical models of bodies in the robot workarea, the controller will warn about a predicted collision and stops the robot if two bodies come too close to each other. The system parameter *Coll-Pred Safety Distance* determines at what distance the two objects are considered to be in collision.

The function *Collision Avoidance* is useful for example when setting up and testing programs, or for programs where positions are not static but created from sensors, such as cameras (non-deterministic programs). By using trigger-signals (see [Trigger signals on page 246](#)), *Collision Avoidance* can be used for implementing safe workspace sharing between multiple robots.

Besides the robot itself the function will monitor up to 10 objects that is created via the configurator in RobotStudio. Typical objects to be monitored are tool mounted on the robot flange, additional equipment mounted on the robot arm (typically axis 3) or static volume around the robot.

The geometric models are set up in RobotStudio.

The functionality is activated by the system input *Collision Avoidance*. A high signal will activate the functionality and a low signal will deactivate the functionality. The functionality is by default active if no signal has been assigned to the system input *Collision Avoidance*.

Collision Avoidance is active both during jogging and when running programs. Also, the RAPID function `IsCollFree` provides a way to check possible collisions before moving to a position.



CAUTION

Always be careful to avoid collisions with external equipment, since a collision could damage the mechanical structure of the arm.

Collision Avoidance is no guarantee for avoiding collisions.



Tip

How to configure *Collision Avoidance* is described in *Operating manual - RobotStudio*.



Tip

Collision Avoidance adds the user configuration in the folder *CA* under the *HOME* folder. This is created when adding a configuration in RobotStudio.

If disk space is needed, the *rsgfx* files can be removed.

Continues on next page

4 Motion Supervision

4.3 Collision Avoidance [3150-1]

Continued

False collision warning

There are different ways to lower the sensitivity of the function *Collision Avoidance* to avoid false warnings.

- Temporarily disable *Collision Avoidance*, see [Disabling Collision Avoidance on page 247](#).
- Decrease the general safety distance with the system parameter *Coll-Pred Safety Distance*.

Activation/deactivation of objects

By default, a defined collision object is active all the time. However, it is possible to configure a collision object with an activation signal, which basically connects it to a digital input that determines whether the object is active or not. This is useful, for example, for modelling multiple tools, where only one tool at a time is active. Another use case is modelling of objects that can be present or absent in the robot cell, for example a pallet.

Note that changing the state of an activation signal will immediately change the activation state of the connected collision object, and no synchronization to the robot path planning is made. Activating a collision object while the robot is moving towards the object can thus lead to a collision because the planned path may already have passed by the collision object while it was inactive. If synchronization is important, then activation signals should either be changed in finepoints when the robot is standing still or using *trigg* instructions like `TriggL` or `TriggJ`.

Trigger signals

A non-moving collision object can be configured with a trigger signal. The value of the trigger signal reflects which robots are in contact with the collision object. More specifically, the value of a trigger signal should be interpreted as a bit pattern, where bit *k* is high if robot *k* is in contact with the collision object. For example, if the trigger signal has the value 6, which is 110 in binary, it means that `ROB_2` and `ROB_3` are in contact with the collision object. Trigger signals can be used to implement safe workspace sharing between multiple robots.

A trigger signal can be configured with two timing behaviors: *immediate* or *on-arrival*. If configured with *immediate* behavior, then the trigger signal is changed as quickly as possible, well before the robot has physically reached the position where it comes into contact with the collision object. If configured with *on-arrival* behavior, then the trigger signal changes state when the robot physically reaches the position where it comes in contact with the zone.

Limitations



CAUTION

Collision Avoidance shall not be used for safety of personnel.

- Paint robots are not supported.
- *Collision Avoidance* cannot be used in manual mode together with responsive jogging. The system parameter *Jog Mode* must be changed to *Standard*.

Continues on next page

- Only stationary/non-moving objects can be configured with a trigger signal. A trigger signal must correspond to a group signal. Furthermore, each collision object must have its own trigger signal.
- There is no support for applications that do corrections to the path, such as conveyor tracking, WeldGuide, Force Control, SoftMove, SoftAct etc.

Disabling *Collision Avoidance*

It is possible to temporarily disable the function *Collision Avoidance* if the robot has already collided or is within the default safety distance, or when the robot arms need to be very close and the risk of collision is acceptable.

Set the system input signal `Collision Avoidance` to 0 to disable *Collision Avoidance*. It is recommended to enable it (set `Collision Avoidance` to 1) as soon as the work is done that required *Collision Avoidance* to be disabled.

4 Motion Supervision

4.4 SafeMove Assistant

4.4 SafeMove Assistant

Purpose

SafeMove Assistant is a functionality in RobotWare that helps users to program their application when there is an active SafeMove configuration. The assistant will read the active configuration and plan the trajectories according to the limits and settings in that configuration. It will set the speed so that SafeMove will not trigger violations etc. It will also stop with error message in case the robot is programmed to enter a forbidden zone etc.

SafeMove Assistant will automatically adjust robot behavior to adopt to the active SafeMove configuration, the robot will adopt to speed limited zones and stop before entering forbidden zones.



CAUTION

SafeMove Assistant is not a safety function.

For example, if using a fence, then a safety distance is required between the safe cartesian zone and the fence.



Note

In case of SafeMove Assistant fails, the SafeMove supervision will trigger an emergency stop.

Description

SafeMove Assistant will check if any SafeMove speed limit is active for any Cartesian speed checkpoint (TCP, tool points, and elbow). If this is the case, a corresponding speed limit is applied in the path planner. For technical reasons, only the speed of the TCP, the wrist center point (WCP), and the elbow are limited by the path planner. Therefore, in cases where other tool points move faster than the TCP, SafeMove may trigger a Tool Speed violation. To avoid this, change the program or decrease the value of the parameter *SafeMove assistance speed factor* (see below).

SafeMove Assistant is not active in manual mode.

SafeMove Assistant does not take path corrections generated at lower level into account. It is therefore an increased risk of SafeMove violations when running applications like Externally Guided Motion or conveyor tracking.

Continues on next page

System parameters

SafeMove Assistant can be disabled for the SafeMove validation etc. This is done with the parameter *Disable SafeMove Assistance*, in the type in *Motion System*.

There are some parameters that can be changed in case robot system has minor overshoot or in any other way triggers SafeMove violations.

Parameter	Description
<i>SafeMove Assistance Speed Factor</i>	That has a default setting of 0.96 which corresponds to 96% of speed supervision will be the speed that path planner will use. This parameter can be decreased to reduce that risk but can in most cases be left at default value.
<i>SafeMove assistance zone margin</i>	When robot is running on a zone border there is a small risk that SafeMove can trigger violations when going in and out of the zone. This parameter can be increased to reduce that risk but can in most cases be left at default value.

For more information, see the parameters in the type *Motion System* described in *Technical reference manual - System parameters*.

This page is intentionally left blank

5 Motor Control

5.1 Independent Axis [3111-1]

5.1.1 Overview

Purpose

The purpose of Independent Axis is to move an axis independently of other axes in the robot system. Some examples of applications are:

- Move an external axis holding an object (for example rotating an object while the robot is spray painting it).
- Save cycle time by performing a robot task at the same time as an external axis performs another.
- Continuously rotate robot axis 6 (for polishing or similar tasks).
- Reset the measurement system after an axis has rotated multiple revolutions in the same direction. Saves cycle time compared to physically winding back.

An axis can move independently if it is set to independent mode. An axis can be changed to independent mode and later back to normal mode again.

What is included

The RobotWare option Independent Axis gives you access to:

- instructions used to set independent mode and specify the movement for an axis
 - an instruction for changing back to normal mode and/or reset the measurement system
 - functions used to verify the status of an independent axis
 - system parameters for configuration.
-

Basic approach

This is the general approach for moving an axis independently. For detailed examples of how this is done, see [Code examples on page 255](#).

- 1 Call an independent move instruction to set the axis to independent mode and move it.
 - 2 Let the robot execute another instruction at the same time as the independent axis moves.
 - 3 When both robot and independent axis has stopped, reset the independent axis to normal mode.
-

Reset axis

Even without being in independent mode, an axis might rotate only in one direction and eventually lose precision. The measurement system can then be reset with the instruction `IndReset`.

The recommendation is to reset the measurement system for an axis before its motor has rotated 10000 revolutions in the same direction.

Continues on next page

5 Motor Control

5.1.1 Overview

Continued

Limitations

A mechanical unit may not be deactivated when one of its axes is in independent mode.

Axes in independent mode cannot be jogged.

The only robot axis that can be used as an independent axis is axis number 6. On IRB 1600, 2600 and 4600 models (except ID version), the instruction `IndReset` can also be used for axis 4.

Internal and customer cabling and equipment may limit the ability to use independent axis functionality on axis 4 and 6.

The option is not possible to use in combination with:

- SafeMove¹
- Track Motion (IRBT)
- Positioners (IRBP) on Interchange axes
- Tool change

¹ *Independent Axis* can in some cases be combined with SafeMove2 if the additional axis does not move the robot, and the additional axis is not monitored by SafeMove. Contact your local ABB sales office team for additional information.

The following is deactivated when option Independent Axes is used:

- Collision detection



Note

The collision detection is deactivated on all axes in a motion planner if one of them is run in independent mode.

5.1.2 System parameters

About the system parameters

This is a brief description of each parameter in the option *Independent Axis*. For more information, see the respective parameter in *Technical reference manual - System parameters*.

Arm

These parameters belongs to the type *Arm* in the topic *Motion*.

Parameter	Description
Independent Joint	Flag that determines if independent mode is allowed for the axis.
Independent Upper Joint Bound	Defines the upper limit of the working area for the joint when operating in independent mode.
Independent Lower Joint Bound	Defines the lower limit of the working area for the joint when operating in independent mode.

Transmission

These parameters belong to the type *Transmission* in the topic *Motion*.

Parameter	Description
Transmission Gear High	Independent Axes requires high resolution in transmission gear ratio, which is therefore defined as <i>Transmission Gear High</i> divided by <i>Transmission Gear Low</i> . If no smaller number can be used, the transmission gear ratio will be correct if <i>Transmission Gear High</i> is set to the number of cogs on the robot axis side, and <i>Transmission Gear Low</i> is set to the number of cogs on the motor side.
Transmission Gear Low	See <i>Transmission Gear High</i> .

5 Motor Control

5.1.3 RAPID components

5.1.3 RAPID components

Data types

There are no data types for Independent Axis.

Instructions

This is a brief description of each instruction in Independent Axis. For more information, see respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

An independent move instruction is executed immediately, even if the axis is being moved at the time. If a new independent move instruction is executed before the last one is finished, the new instruction immediately overrides the old one.

Instruction	Description
IndAMove	IndAMove (Independent Absolute position Movement) change an axis to independent mode and move the axis to a specified position.
IndCMove	IndCMove (Independent Continuous Movement) change an axis to independent mode and start moving the axis continuously at a specified speed.
IndDMove	IndDMove (Independent Delta position Movement) change an axis to independent mode and move the axis a specified distance.
IndRMove	IndRMove (Independent Relative position Movement) change a rotational axis to independent mode and move the axis to a specific position within one revolution. Because the revolution information in the position is omitted, IndRMove never rotates more than one axis revolution.
IndReset	IndReset is used to change an independent axis back to normal mode. IndReset can move the measurement system for a rotational axis a number of axis revolutions. The resolution of positions is decreased when moving away from logical position 0, and winding the axis back would take time. By moving the measurement system the resolution is maintained without physically winding the axis back. Both the independent axis and the robot must stand still when calling IndReset.

Functions

This is a brief description of each function in Independent Axis. For more information, see respective function in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
IndInpos	IndInpos indicates whether an axis has reached the selected position.
IndSpeed	IndSpeed indicates whether an axis has reached the selected speed.

5.1.4 Code examples

Save cycle time

An object in station A needs welding in two places. The external axis for station A can turn the object in position for the second welding while the robot is welding on another object. This saves cycle time compared to letting the robot wait while the external axis moves.

```

!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;

```

Polish by rotating axis 6

To polish an object the robot axis 6 can be set to continuously rotate.

Set robot axis 6 to independent mode and continuously rotate it. Move the robot over the area you want to polish. Stop movement for both robot and independent axis before changing back to normal mode. After rotating the axis many revolutions, reset the measurement system to maintain the resolution.

Note that, for this example to work, the parameter *Independent Joint* for rob1_6 must be set to 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;

```

Continues on next page

5 Motor Control

5.1.4 Code examples

Continued

```
!Stop axis 6 and wait until it's still
IndCMove ROB_1, 6, 0;
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
```

Reset an axis

This is an example of how to reset the measurement system for axis 1 in station A. The measurement system will change a whole number of revolutions, so it is close to zero ($\pm 180^\circ$).

```
IndReset Station_A, 1 \RefNum:=0 \Short;
```

6 RAPID Program Features

6.1 Path Recovery [3113-1]

6.1.1 Overview

Purpose

Path Recovery is used to store the current movement path, perform some robot movements and then restore the interrupted path. This is useful when an error or interrupt occurs during the path movement. An error handler or interrupt routine can perform a task and then recreate the path.

For applications like arc welding and gluing, it is important to continue the work from the point where the robot left off. If the robot started over from the beginning, then the work piece would have to be scrapped.

If a process error occurs when the robot is inside a work piece, moving the robot straight out might cause a collision. By using the path recorder, the robot can instead move out along the same path it came in.

What is included

The RobotWare option Path Recovery gives you access to:

- instructions to suspend and resume the coordinated synchronized movement mode on the error or interrupt level.
- a path recorder, with the ability to move the TCP out from a position along the same path it came.

Limitations

The instructions `StorePath` and `RestoPath` only handles movement path data. The stop position must also be stored.

Movements using the path recorder has to be performed on trap-level, i.e. `StorePath` has to be executed prior to `PathRecMoveBwd`.

6 RAPID Program Features

6.1.2 RAPID components

6.1.2 RAPID components

Data types

This is a brief description of each data type in Path Recovery. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
pathrecid	pathrecid is used to identify a breakpoint for the path recorder.

Instructions

This is a brief description of each instruction in Path Recovery. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
StorePath	StorePath is used to store the movement path being executed when an error or interrupt occurs.
RestoPath	RestoPath is used to restore the path that was stored by StorePath.
PathRecStart	PathRecStart is used to start recording the robot's path. The path recorder will store path information during execution of the robot program.
PathRecStop	PathRecStop is used to stop recording the robot's path.
PathRecMoveBwd	PathRecMoveBwd is used to move the robot backwards along a recorded path.
PathRecMoveFwd	PathRecMoveFwd is used to move the robot back to the position where PathRecMoveBwd was executed. It is also possible to move the robot partly forward by supplying an identifier that has been passed during the backward movement.
SyncMoveSuspend	SyncMoveSuspend is used to suspend synchronized movements mode and set the system to independent movement mode.
SyncMoveResume	SyncmoveResume is used to go back to synchronized movements from independent movement mode.

Functions

This is a brief description of each function in Path Recovery. For more information, see the respective function in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
PathRecValidBwd	PathRecValidBwd is used to check if the path recorder is active and if a recorded backward path is available.
PathRecValidFwd	PathRecValidFwd is used to check if the path recorder can be used to move forward. The ability to move forward with the path recorder implies that the path recorder must have been ordered to move backwards earlier.

6.1.3 Store current path

Why store the path?

The simplest way to use Path Recovery is to only store the current path to be able to restore it after resolving an error or similar action.

Let's say that an error occur during arc welding. To resolve the error the robot might have to be moved away from the part. When the error is resolved, the welding should be continued from the point it left off. This is solved by storing the path information and the position of the robot before moving away from the path. The path can then be restored and the welding resumed after the error has been handled.

Basic approach

This is the general approach for storing the current path:

- 1 At the start of an error handler or interrupt routine:
 - stop the movement
 - store the movement path
 - store the stop position
- 2 At the end of the error handler or interrupt routine:
 - move to the stored stop position
 - restore the movement path
 - start the movement

Example

This is an example of how to use Path Recovery in error handling. First the path and position is stored, the error is corrected and then the robot is moved back in position and the path is restored.

```

MoveL p100, v100, z10, gun1;
...
ERROR
  IF ERRNO=MY_GUN_ERR THEN
    gun_cleaning();
  ENDIF
...
PROC gun_cleaning()
  VAR robtargt p1;

  !Stop the robot movement, if not already stopped.
  StopMove;

  !Store the movement path and current position
  StorePath;
  p1 := CRobT(\Tool:=gun1\WObj:=wobj0);

  !Correct the error
  MoveL pclean, v100, fine, gun1;

```

Continues on next page

6 RAPID Program Features

6.1.3 Store current path

Continued

```
...  
!Move the robot back to the stored position  
MoveL p1, v100, fine, gun1;  
  
!Restore the path and start the movement  
RestoPath;  
StartMove;  
RETRY;  
ENDPROC
```

6.1.4 Path recorder

What is the path recorder

The path recorder can memorize a number of move instructions. This memory can then be used to move the robot backwards along that same path.

How to use the path recorder

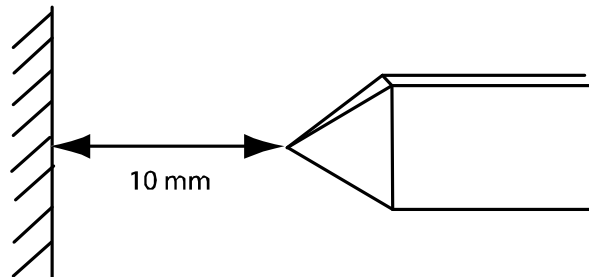
This is the general approach for using the path recorder:

- 1 Start the path recorder
- 2 Move the robot with regular move, or process, instructions
- 3 Store the current path
- 4 Move backwards along the recorded path
- 5 Resolve the error
- 6 Move forward along the recorded path
- 7 Restore the interrupted path

Lift the tool

When the robot moves backward in its own track, you may want to avoid scraping the tool against the work piece. For a process like arc welding, you want to stay clear of the welding seam.

By using the argument `ToolOffs` in the instructions `PathRecMoveBwd` and `PathRecMoveFwd`, you can set an offset for the TCP. This offset is set in tool coordinates, which means that if it is set to `[0,0,10]` the tool will be 10mm from the work object when it moves back along the recorded path.



xx0400000828



Note

When a MultiMove system is in synchronized mode all tasks must use `ToolOffs` if a tool is going to be lifted.

However if you only want to lift one tool, set `ToolOffs=[0,0,0]` in the other tasks.

Simple example

If an error occurs between p1 and p4, the robot will return to p1 where the error can be resolved. When the error has been resolved, the robot continues from where the error occurred.

Continues on next page

6 RAPID Program Features

6.1.4 Path recorder

Continued

When p4 is reached without any error, the path recorder is switched off. The robot then moves from p4 to p5 without the path recorder.

```
...
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
...
```

Complex example

In this example, the path recorder is used for two purposes:

- If an error occurs, the operator can choose to back up to p1 or to p2. When the error has been resolved, the interrupted movement is resumed.
- Even if no error occurs, the path recorder is used to move the robot from p4 to p1. This technique is useful when the robot is in a narrow position that is difficult to move out of.

Note that if an error occurs during the first move instruction, between p1 and p2, it is not possible to go backwards to p2. If the operator choose to go back to p2, `PathRecValidBwd` is used to see if it is possible. Before the robot is moved forward to the position where it was interrupted, `PathRecValidFwd` is used to see if it is possible (if the robot never backed up it is already in position).

```
...
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
```

Continues on next page

```
StorePath;
PathRecMoveBwd \ID:=origin_id
    \ToolOffs:=[0,0,10];
RestoPath;
PathRecStop \Clear;
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;
...
```

Resume path recorder

If the path recorder is stopped, it can be started again from the same position without losing its history.

In the example below, the `PathRecMoveBwd` instruction will back the robot to p1. If the robot had been in any other position than p2 when the path recorder was restarted, this would not have been possible.

Continues on next page

6 RAPID Program Features

6.1.4 Path recorder

Continued

For more information, see the section about `PathRecStop` in *Technical reference manual - RAPID Instructions, Functions and Data types*.

```
...
MoveL p1, vmax, z50, tool1;
PathRecStart id1;
MoveL p2, vmax, z50, tool1;
PathRecStop;
MoveL p3, vmax, z50, tool1;
MoveL p4, vmax, z50, tool1;
MoveL p2, vmax, z50, tool1;
PathRecStart id2;
MoveL p5, vmax, z50, tool1;
StorePath;
PathRecMoveBwd \ID:=id1;
RestoPath;
...
```


6.2 Multitasking [3114-1]

6.2.1 Introduction to Multitasking

Purpose

The purpose of the option *Multitasking* is to be able to execute more than one program at a time.

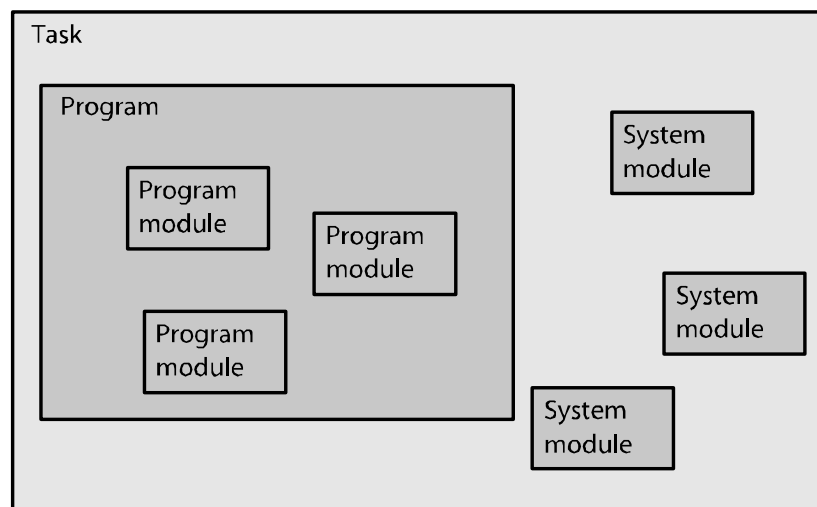
Examples of applications to run in parallel with the main program:

- Continuous supervision of signals, even if the main program has stopped. This can in some cases take over the job of a PLC. However, the response time will not match that of a PLC.
- Operator input from the FlexPendant while the robot is working.
- Control and activation/deactivation of external equipment.

Basic description

Up to 20 tasks can be run at the same time.

Each task consists of one program (with several program modules) and several system modules. The modules are local in the respective task.



en0300000517

Variables and constants are local in the respective task, but persistents are not. Every task has its own trap handling and event routines are triggered only on its own task system states.

What is included

The RobotWare option Multitasking gives you access to:

- The possibility to run up to 20 programs in parallel (one per task).
- The system parameters: The type *Task* and all its parameters.
- The data types: *taskid*, *syncident*, and *tasks*.
- The instruction: *WaitSyncTask*.
- The functions: *TestAndSet*, *TaskRunMec*, and *TaskRunRob*.

Continues on next page

6 RAPID Program Features

6.2.1 Introduction to Multitasking

Continued



Note

`TestAndSet`, `TaskRunMec`, and `TaskRunRob` can be used without the option `Multitasking`, but they are much more useful together with `Multitasking`.

Basic approach

This is the basic approach for setting up Multitasking. For more information, see [RAPID components on page 269](#).

- 1 Define the tasks you need.
- 2 Write RAPID code for each task.
- 3 Specify which modules to load in each task.

6.2.2 System parameters

About the system parameters

This is a brief description of each parameter in the option *Multitasking*. For more information, see the respective parameter in *Technical reference manual - System parameters*.

Task

These parameters belongs to the type *Task* in the topic *Controller*.

Parameter	Description
Task	<p>The name of the task.</p> <p>Note that the name of the task must be unique. This means that it cannot have the same name as the mechanical unit, and no variable in the RAPID program can have the same name.</p> <p>Note that editing the task entry in the configuration editor and changing the task name will remove the old task and add a new one. This means that any program or module in the task will disappear after a restart with these kind of changes.</p>
Task in foreground	<p>Used to set priorities between tasks.</p> <p><i>Task in foreground</i> contains the name of the task that should run in the foreground of this task. This means that the program of the task, for which the parameter is set, will only execute if the foreground task program is idle.</p> <p>If <i>Task in foreground</i> is set to empty string for a task, it runs at the highest level.</p>
Type	<p>Controls the start/stop and system restart behavior:</p> <ul style="list-style-type: none"> • Normal (NORMAL) - The task program is manually started and stopped (e.g. from the FlexPendant). The task stops at emergency stop. • Static (STATIC) - At a restart the task program continues from where the it was. The task program is normally not stopped by the FlexPendant or by emergency stop. • Semistatic (SEMISTATIC) - The task program restarts from the beginning at restart. The task program is normally not stopped by the FlexPendant or by emergency stop. <p>A task that controls a mechanical unit must be of the type <i>normal</i>.</p>
Main entry	The name of the start routine for the task program.
Check unresolved references	This parameter should be set to NO if the system is to accept unsolved references in the program while linking a module, otherwise set to YES.
TrustLevel	<p><i>TrustLevel</i> defines the system behavior when a static or semistatic task program is stopped (e.g. due to error):</p> <ul style="list-style-type: none"> • SysFail - If the program of this task stops, the system will be set to SYS_FAIL. This will cause the programs of all NORMAL tasks to stop (static and semistatic tasks will continue execution if possible). No jogging or program start can be made. A restart is required. • SysHalt - If the program of this task stops, the programs of all normal tasks will be stopped. If "motors on" is set, jogging is possible, but not program start. A restart is required. • SysStop - If the program of this task stops, the programs of all normal tasks will be stopped but are restartable. Jogging is also possible. • NoSafety - Only the program of this task will stop.

Continues on next page

6 RAPID Program Features

6.2.2 System parameters

Continued

Parameter	Description
MotionTask	Indicates whether the task program can control robot movement with RAPID move instructions. Only one task can have <i>MotionTask</i> set to YES unless the option MultiMove is used.

6.2.3 RAPID components

Data types

This is a brief description of each data type in Multitasking. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
taskid	taskid identify available tasks in the system. This identity is defined by the system parameter <i>Task</i> , and cannot be defined in the RAPID program. However, the data type <i>taskid</i> can be used as a parameter when declaring a routine. For code example, see taskid on page 280 .
syncident	syncident is used to identify the waiting point in the program, when using the instruction <i>WaitSyncTask</i> . The name of the <i>syncident</i> variable must be the same in all task programs. For code example, see WaitSyncTask example on page 274 .
tasks	A variable of the data type <i>tasks</i> contains names of the tasks that will be synchronized by the instruction <i>WaitSyncTask</i> . For code example, see WaitSyncTask example on page 274 .

Instructions

This is a brief description of each instruction in Multitasking. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
WaitSyncTask	<i>WaitSyncTask</i> is used to synchronize several task programs at a special point in the program. A <i>WaitSyncTask</i> instruction will delay program execution and wait for the other task programs. When all task programs have reached the point, the respective program will continue its execution. For code example, see WaitSyncTask example on page 274 .

Functions

This is a brief description of each function in Multitasking. For more information, see the respective function in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
TestAndSet	<i>TestAndSet</i> is used, together with a boolean flag, to ensure that only one task program at the time use a specific RAPID code area or system resource. For code example, see Example with flag and TestAndSet on page 278 .
TaskRunMec	Check if the task program controls any mechanical unit (robot or other unit). For code example, see Test if task controls mechanical unit on page 279 .
TaskRunRob	Check if the task program controls any robot with TCP. For code example, see Test if task controls mechanical unit on page 279 .

6 RAPID Program Features

6.2.4.1 Persistent variables

6.2.4 Communication between tasks

6.2.4.1 Persistent variables

About persistent variables

To share data between tasks, use persistent variables.

A persistent variable is global in all tasks where it is declared. The persistent variable must be declared as the same type and size (array dimension) in all tasks. Otherwise a runtime error will occur.

It is sufficient to specify an initial value for the persistent variable in one task. If initial values are specified in several tasks, only the initial value of the first module to load will be used.



Tip

When a program is saved, the current value of a persistent variable will be used as initial value in the future. If this is not desired, reset the persistent variable directly after the communication.

Example with persistent variable

In this example the persistent variables `startsync` and `stringtosend` are accessed by both tasks, and can therefore be used for communication between the task programs.

Main task program:

```
MODULE module1
  PERS bool startsync:=FALSE;
  PERS string stringtosend:="";
  PROC main()
    stringtosend:="this is a test";
    startsync:= TRUE
  ENDPROC
ENDMODULE
```

Background task program:

```
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
```

Continues on next page

Module for common data

When using persistent variables in several tasks, there should be declarations in all the tasks. The best way to do this, to avoid type errors or forgetting a declaration somewhere, is to declare all common variables in a system module. The system module can then be loaded into all tasks that require the variables.

6 RAPID Program Features

6.2.4.2 Waiting for other tasks

6.2.4.2 Waiting for other tasks

Two techniques

Some applications have task programs that execute independently of other tasks, but often task programs need to know what other tasks are doing.

A task program can be made to wait for another task program. This is accomplished either by setting a persistent variable that the other task program can poll, or by setting a signal that the other task program can connect to an interrupt.

Polling

This is the easiest way to make a task program wait for another, but the performance will be the slowest. Persistent variables are used together with the instructions `WaitUntil` or `WHILE`.

If the instruction `WaitUntil` is used, it will poll internally every 100 ms.



CAUTION

Do not poll more frequently than every 100 ms. A loop that polls without a wait instruction can cause overload, resulting in lost contact with the FlexPendant.

Polling example

Main task program:

```
MODULE module1
  PERS bool startsync:=FALSE;
  PROC main()
    startsync:= TRUE;
    ...
  ENDPROC
ENDMODULE
```

Background task program:

```
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
```

Interrupt

By setting a signal in one task program and using an interrupt in another task program, quick response is obtained without the work load caused by polling.

The drawback is that the code executed after the interrupt must be placed in a trap routine.

Continues on next page

Interrupt example

Main task program:

```
MODULE module1
  PROC main()
    SetDO d01,1;
    ...
  ENDPROC
ENDMODULE
```

Background task program:

```
MODULE module2
  VAR intnum intn01;

  PROC main()
    CONNECT intn01 WITH wait_trap;
    ISignalDO d01, 1, intn01;
    WHILE TRUE DO
      WaitTime 10;
    ENDWHILE
  ENDPROC

  TRAP wait_trap
    ! This is the point where the execution
    ! continues after d01 is set in main task
    ...
    IDelete intn01;
  ENDTRAP
ENDMODULE
```

6 RAPID Program Features

6.2.4.3 Synchronizing between tasks

6.2.4.3 Synchronizing between tasks

Synchronizing using WaitSyncTask

Synchronization is useful when task programs are depending on each other. No task program will continue beyond a synchronization point in the program code until all task programs have reached that point in the respective program code.

The instruction `WaitSyncTask` is used to synchronize task programs. No task program will continue its execution until all task programs have reached the same `WaitSyncTask` instruction.

WaitSyncTask example

In this example, the background task program calculates the next object's position while the main task program handles the robots work with the current object.

The background task program may have to wait for operator input or I/O signals, but the main task program will not continue with the next object until the new position is calculated. Likewise, the background task program must not start the next calculation until the main task program is done with one object and ready to receive the new value.

Main task program:

```
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
```

Background task program:

```
MODULE module2
  PERS pos object_position:= [0,0,0];
  PERS tasks task_list{2} := [{"MAIN"}, {"BACK1"}];
  VAR syncident sync1;
```

Continues on next page

```
PROC main()
  WHILE TRUE DO
    !Call routine to calculate object_position
    calculate_position;

    !Wait for handling of current object
    WaitSyncTask sync1, task_list;
  ENDWHILE
ENDPROC

PROC calculate_position()
  ...
  object_position:= ...
ENDPROC
ENDMODULE
```

6 RAPID Program Features

6.2.4.4 Using a dispatcher

6.2.4.4 Using a dispatcher

What is a dispatcher?

A digital signal can be used to indicate when another task should do something. However, it cannot contain information about what to do.

Instead of using one signal for each routine, a dispatcher can be used to determine which routine to call. A dispatcher can be a persistent string variable containing the name of the routine to execute in another task.

Dispatcher example

In this example, the main task program calls routines in the background task by setting `routine_string` to the routine name and then setting `do5` to 1. In this way, the main task program initialize that the background task program should execute the routine `clean_gun` first and then `routine1`.

Main task program:

```
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
```

Background task program:

```
MODULE module2
  PERS string routine_string:="";

  PROC main()
    WaitDO do5,1;
    %routine_string%;
    SetDO do5,0;
  ENDPROC

  PROC clean_gun()
    ...
  ENDPROC

  PROC routine1()
    ...
```

Continues on next page

```
ENDPROC  
ENDMODULE
```

6 RAPID Program Features

6.2.5.1 Share resource between tasks

6.2.5 Other programming issues

6.2.5.1 Share resource between tasks

Flag indicating occupied resource

System resources, such as FlexPendant, file system and I/O signals, are available from all tasks. However, if several task programs use the same resource, make sure that they take turns using the resource, rather than using it at the same time.

To avoid having two task programs using the same resource at the same time, use a flag to indicate that the resource is already in use. A boolean variable can be set to true while the task program uses the resource.

To facilitate this handling, the instruction `TestAndSet` is used. It will first test the flag. If the flag is false, it will set the flag to true and return true. Otherwise, it will return false.

Example with flag and `TestAndSet`

In this example, two task programs try to write three lines each to the FlexPendant. If no flag is used, there is a risk that these lines are mixed with each other. By using a flag, the task program that first execute the `TestAndSet` instruction will write all three lines first. The other task program will wait until the flag is set to false and then write all its lines.

Main task program:

```
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;
```

Background task program:

```
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;
```

6.2.5.2 Test if task controls mechanical unit

Two functions for inquiring

There are functions for checking if the task program has control of any mechanical unit, `TaskRunMec`, or of a robot, `TaskRunRob`.

`TaskRunMec` will return true if the task program controls a robot or other mechanical unit. `TaskRunRob` will only return true if the task program controls a robot with TCP.

`TaskRunMec` and `TaskRunRob` are useful when using `MultiMove`. With `MultiMove` you can have several tasks controlling mechanical units, see *Application manual - MultiMove*.



Note

For a task to have control of a robot, the parameter *Type* must be set to normal, and the type *MotionTask* must be set to YES. See [System parameters on page 267](#).

Example with `TaskRunMec` and `TaskRunRob`

In this example, the maximum speed for external equipment is set. If the task program controls a robot, the maximum speed for external equipment is set to the same value as the maximum speed for the robot. If the task program controls external equipment but no robot, the maximum speed is set to 5000 mm/s.

```
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
```

6 RAPID Program Features

6.2.5.3 taskid

6.2.5.3 taskid

taskid syntax

A task always has a predefined variable of type taskid that consists of the name of the task and the suffix "Id". For example, the variable name of the MAIN task is MAINId.

Code example

In this example, the module PART_A is saved in the task BACK1, even though the Save instruction is executed in another task.

BACK1Id is a variable of type taskid that is automatically declared by the system.

```
Save \TaskRef:=BACK1Id, "PART_A"  
  \FilePath:="HOME:/DOORDIR/PART_A.MOD";
```


6.2.5.4 Avoid heavy loops

Background tasks loop continuously

A task program is normally executed continuously. This means that a background task program is in effect an eternal loop. If this program does not have any waiting instruction, the background task may use too much computer power and make the controller unable to handle the other tasks.

Example

```
MODULE background_module
  PROC main()
    WaitTime 1;
    IF di1=1 THEN
      . . .
    ENDIF
  ENDPROC
ENDMODULE
```

If there was no wait instruction in this example and `di1` was 0, then this background task would use up the computer power with a loop doing nothing.

This page is intentionally left blank

7 Communication

7.1 FTP&SFTP client [3116-1]

7.1.1 Introduction to FTP&SFTP client

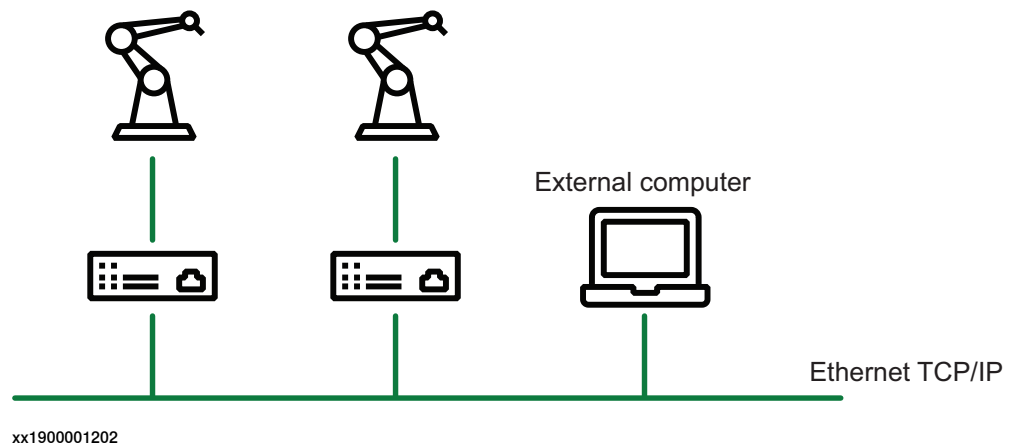
Purpose

The purpose of FTP&SFTP Client is to enable the robot to access remote mounted disks, for example a hard disk drive on a PC.

Here are some examples of applications:

- Backup to a remote computer.
- Load programs from a remote computer.

Network illustration



Description

Several robots can access the same computer over an Ethernet network.

The FTP/SFTP mounted device is accessed by its name, as specified in the system parameter *Name*.

Once the FTP/SFTP protocol is configured, the remote computer can be accessed in the same way as the controller's internal hard disk.

What is included

The RobotWare option *FTP&SFTP client* gives you access to the system parameter types *FTP Client* and *SFTP Client*.

Basic approach

This is the general approach for using FTP&SFTP client.

- 1 Configure an FTP/SFTP protocol to point out a disk or directory on a remote computer that will be accessible from the robot.
- 2 Read and write to the remote computer in the same way as with the controller's internal hard disk.

Continues on next page

7 Communication

7.1.1 Introduction to FTP&SFTP client

Continued

SFTP supports the following servers:

- Rebox version 1.0.3
- CompleteFTP version 11.0.0
- Cerberus version 9.0.4.0

In certain SFTP servers, as Complete SFTP server, there is a configuration setting, **Timeout for idle sessions**, which defines the time that the connection can be idle. If no client requests are made during this time interval, the connection is closed. Setting the value as **No timeout** will keep the connection alive, even though client requests are not made.

Requirements

The external computer must have:

- TCP/IP stack
- FTP Server or SFTP Server

Directory listing style on FTP server

The FTP server must list directories in a UNIX style.

Example:

```
drwxrwxrwx 1 owner group 25 May 18 16:39 backups
```

The MS-DOS style does not work.



Tip

For Internet Information Services (IIS) in Windows, the directory listing style is configurable.

Limitations

- The maximum length of the path is 248 characters. Note that for backup and restore, the names of all the created directories have to be included in the max path.

System parameters

See *Technical reference manual - System parameters*.

7.2 NFS Client [3117-1]

7.2.1 Introduction to NFS Client

Purpose

The purpose of NFS Client is to enable the robot to access remote mounted disks, for example a hard disk drive on a PC.

Here are some examples of applications:

- Backup to a remote computer.
- Load programs from a remote computer.

Description

Several robots can access the same computer over an Ethernet network.

The NFS mounted device is accessed by its name, as specified in the `Name` system parameter.

Once the NFS application protocol is configured, the remote computer can be accessed in the same way as the controller's internal hard disk.

What is included

The RobotWare option *NFS Client* gives you access to the system parameter type *Application protocol* and its parameters: *Name*, *Type*, *Transmission protocol*, *Server address*, *Server type*, *Trusted*, *Local path*, *Server path*, *User ID*, *Group ID*, and *Show Device*.

Basic approach

This is the general approach for using NFS Client.

- 1 Configure an NFS protocol to point out a disk or directory on a remote computer that will be accessible from the robot.
- 2 Read and write to the remote computer in the same way as with the controller's internal hard disk.

Prerequisites

The external computer must have:

- TCP/IP stack
- NFS Server

Limitations

When using the NFS Client the maximum length for a file path including the file name is 248 characters. The whole path is included in the 248 characters, not only the server path. When ordering a backup towards a mounted disk all the directories created by the backup has to be included in the max path.

This page is intentionally left blank

8 User Interaction Application

8.1 RobotStudio Connect [3119-1]

Overview

RobotStudio is the programming, configuration and commissioning tool for OmniCore controllers. RobotStudio acts directly on the active data in the controller and enables activities like RAPID programming, update/booting of the systems software and system configuration. Connecting RobotStudio directly to the local management port is enabled by default, but connecting RobotStudio over a public network requires this option *RobotStudio Connect*.

8 User Interaction Application

8.2 FlexPendant Base Apps

8.2 FlexPendant Base Apps

Limited App Package [3120-1]

The option *Limited App Package* contains base functionality to operate the robot system. This base version of software for the FlexPendant allows for the most crucial functionality, like jogging the robot, calibration of the robot, basic operation (start, stop, loading programs), read and write I/O signals, event log and operator messages.

Essential App Package [3120-2]

The option *Essential App Package* includes features that will make it easier and more efficient to work with the robot system. The jog functionality is improved with 3D illustrations, and dashboards makes it easy to view the system status at a glance. This includes the option *Limited App Package*.

8.3 FlexPendant Independent Apps

Program Package [3151-1]

The option *Program Package* is required in order to create new and edit existing RAPID programs on the FlexPendant. If the program package is not selected with the FlexPendant, RobotStudio must instead be used on a separate PC to create and edit RAPID programs.

The FlexPendant options are not tied to the FlexPendant hardware, but instead to OmniCore controller. This means a FlexPendant runs the apps that are licensed to the controller it is connected to. A shared FlexPendant can accordingly have different apps on different robots.

This page is intentionally left blank

9 Engineering tools

9.1 RobotWare Add-In

Required for licensed Add-Ins.

9 Engineering tools

9.2.1 Overview

9.2 Path Corrections [3123-1]

9.2.1 Overview

Purpose

The purpose of Path Corrections is to be able to make online adjustments of the robot path according to input from sensors. With the set of instructions that Path Corrections offers, the robot path can be compared and adjusted with the input from sensors.

What is included

The RobotWare option Path Corrections gives you access to:

- the data type `corrdescr`
 - the instructions `CorrCon`, `CorrDiscon`, `CorrClear` and `CorrWrite`
 - the function `CorrRead`
-

Basic approach

This is the general approach for setting up Path Corrections. For a detailed example of how this is done, see [Code example on page 296](#).

- 1 Declare the correction generator.
- 2 Connect the correction generator.
- 3 Define a trap routine that determines the offset and writes it to the correction generator.
- 4 Define an interrupt to frequently call the trap routine.
- 5 Call a move instruction using the correction. The path will be repeatedly corrected.



Note

The instruction `CorrWrite` is intended with low speed and moderate values of correction. Too aggressive values will be clamped. The correction values should be tested in RobotStudio to confirm the performance.



Note

If two or more move instructions are called after each other with the `\Corr` switch, it is important to know that all `\Corr` offsets are reset each time the robot starts from a finepoint. So, when using finepoints, on the second `Move` instruction the controller does not know that the path already has an offset. To avoid any strange behavior it is recommended only to use zones together with the `\Corr` switch and avoid finepoints.

Continues on next page

Limitations

It is possible to connect several correction generators at the same time (for instance one for corrections along the Z axis and one for corrections along the Y axis). However, it is not possible to connect more than 5 correction generators at the same time.

After a controller restart, the correction generators have to be defined once again. The definitions and connections do not survive a controller restart.

The instructions can only be used in motion tasks.

9 Engineering tools

9.2.2 RAPID components

9.2.2 RAPID components

Data types

This is a brief description of each data type in the option *Path Corrections*. For more information, see the respective data type in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Data type	Description
corrdescr	corrdescr is a correction generator descriptor that is used as the reference to the correction generator.

Instructions

This is a brief description of each instruction in the option *Path Corrections*. For more information, see the respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instruction	Description
CorrCon	CorrCon activates path correction. Calling CorrCon will connect a correction generator. Once this connection is made, the path can be continuously corrected with new offset inputs (for instance from a sensor).
CorrDiscon	CorrDiscon deactivates path correction. Calling CorrDiscon will disconnect a correction generator.
CorrClear	CorrClear deactivate path correction. Calling CorrClear will disconnect all correction generators.
CorrWrite	CorrWrite sets the path correction values. Calling CorrWrite will set the offset values to a correction generator.

Functions

This is a brief description of each function in the option *Path Corrections*. For more information, see the respective function in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Function	Description
CorrRead	CorrRead reads the total correction made by a correction generator.

9.2.3 Related RAPID functionality

The argument \Corr

The optional argument \Corr can be set for some move instructions. This will enable path corrections while the move instruction is executed.

The following instructions have the optional argument \Corr:

- MoveL
- MoveC
- SearchL
- SearchC
- TriggL (only if the controller is equipped with the base functionality Fixed Position Events)
- TriggC (only if the controller is equipped with the base functionality Fixed Position Events)
- CapL (only if the controller is equipped with the option Continuous Application Platform)
- CapC (only if the controller is equipped with the option Continuous Application Platform)
- ArcL (only if the controller is equipped with the option RobotWare Arc)
- ArcC (only if the controller is equipped with the option RobotWare Arc)

For more information on these instructions, see respective instruction in *Technical reference manual - RAPID Instructions, Functions and Data types*.

Interrupts

To create programs using Path Corrections, you need to be able to handle interrupts. For more information on interrupts, see *Technical reference manual - RAPID Overview*.

9 Engineering tools

9.2.4 Code example

9.2.4 Code example

Linear movement with correction

This is a simple example of how to program a linear path with online path correction. This is done by having an interrupt 5 times per second, calling a trap routine which makes the offset correction.

Program code

```
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
```


9.3 Auto Acknowledge Input

Description

The RobotWare base functionality *Auto Acknowledge Input* is an option that enables a system input which will acknowledge the dialog presented on the FlexPendant when switching the operator mode from manual to auto with the key switch on the robot controller.



WARNING

Note that using such an input will be contrary to the regulations in the safety standard ISO 10218-1 chapter 5.3.5 Single point of control with following text:

"The robot control system shall be designed and constructed so that when the robot is placed under local pendant control or other teaching device control, initiation of robot motion or change of local control selection from any other source shall be prevented."

Thus it is absolutely necessary to use other means of safety to maintain the requirements of the standard and the machinery directive and also to make a risk assessment of the completed cell. Such additional arrangements and risk assessment is the responsibility of the system integrator and the system must not be put into service until these actions have been completed.

Limitations

The system parameter cannot be defined using the FlexPendant or RobotStudio, only with a text string in the configuration file.

Activate Auto Acknowledge Input

The robot system must be installed with the option *Auto Acknowledge Input* using the **Modify Installation** function.

Use the following procedure to activate the system input for *Auto Acknowledge Input*.

	Action
1	Save a copy of the configuration file <i>sys.cfg</i> , using the FlexPendant or RobotStudio.
2	Edit the configuration file <i>sys.cfg</i> , using a text editor. Add the following line in the group SYSSIG_IN: <pre>-Name "my_signal_name" -Action "AckAutoMode"</pre> <i>my_signal_name</i> is the name of the configured digital input signal that should be used as the system input.
3	Save the file and reload it to the controller.
4	Restart the system to activate the signal.

This page is intentionally left blank

10 Tool control options

10.1 Servo Tool Change [3110-1]

10.1.1 Overview

Purpose

The purpose of Servo Tool Change is to be able to change tools online.

With the option Servo Tool Change it is possible to disconnect the cables to the motor of an additional axis and connect them to the motor of another additional axis. This can be done on the run, in production.

This option is designed with servo tools in mind, but can be used for any type of additional axes.

Examples of advantages are:

- One robot can handle several tools.
- Less equipment is needed since one drive-measurement system is shared by several tools.

What is included

The RobotWare option *Servo Tool Change* enables:

- changing tool online
- up to 8 different servo tools to change between.

Note that the option Servo Tool Change only provides the software functionality. Hardware, such as a tool changer is not included.

Basic approach

This is the general approach for using Servo Tool Change. For a more detailed description of how this is done, see [Tool change procedure on page 305](#).

- 1 Deactivate the first tool.
- 2 Disconnect the first tool from the cables.
- 3 Connect the second tool to the cables.
- 4 Activate the second tool.

10 Tool control options

10.1.2 Requirements and limitations

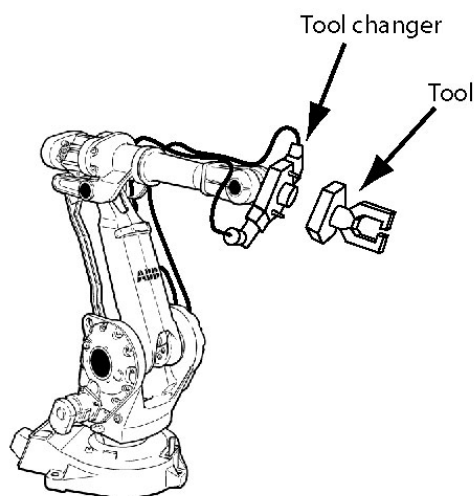
10.1.2 Requirements and limitations

Additional axes

To use Servo Motor Control, you must have the option Additional Axes. All additional axes used by servo motor control must be configured according to the instructions in *Application manual - Additional axes*.

Tool changer

To be able to change tools in production with a plug-in mechanism, a mechanical tool changer interface is required.



en0300000549

All cables are connected to the tool changer. The tool changer interface includes connections for signals, power, air, water, or whatever needs to be transmitted to and from the tool.

Up to 8 tools

Up to 8 additional axes (servo tools or other axes) can be installed simultaneously in one robot controller. Some of them (or all) may be servo tools sharing a tool changer.

Moving deactivated tool

The controller remembers the position of a deactivated tool. When the tool is reconnected and activated this position is used.

If the servo tool axis is moved during deactivation, the position of the axis might be wrong after activation, and this will not be detected by the controller.

Continues on next page

The position after activation will be correct if the axis has not been moved, or if the movement is less than 0.5 motor revolutions.



Tip

If you have the Spot Servo option you can use tool change calibration.

After a tool is activated, use the instruction `STCalib` to calibrate the tool. This will adjust any positional error caused by tool movements during deactivation.

Activating wrong tool

It is important to only activate a mechanical unit that is connected.

An activation of the wrong mechanical unit may cause unexpected movements or errors. The same errors occur if a tool is activated when no tool at all is connected.



Tip

A connection relay can be configured so that activation of a mechanical unit is only allowed when it is connected. See [Connection relay on page 303](#).

10 Tool control options

10.1.3 Configuration

10.1.3 Configuration

Configuration overview

The option Servo Tool Change allows configuration of several tools for the same additional axis.

One individual set of parameters is installed for each gun tool.

How to configure each tool

Each tool is configured the same way as if it was the only tool. For information on how to do this, see *Application manual - Additional axes*.

The parameter *Deactivate PTC superv. at disconnect*, in the type *Mechanical Unit*, must be set to Yes.

The parameter *Disconnect at Deactivate*, in the type *Measurement Channel*, must be set to Yes.

The parameter *Logical Axis*, in the type *Joint*, can be set to the same number for several tools. Since the tools are never used at the same time, the tools are allowed to use the same logical axis.

The parameter *allow_activation_from_any_motion_task*, in the type *Mechanical Unit*, must be set for the specific servo gun. The servo gun .cfg files are created by the servo gun manufacturer.

For a detailed description of the respective parameter, see *Technical reference manual - System parameters*.

10.1.4 Connection relay

Overview

To make sure a disconnected mechanical unit is not activated, a connection relay can be used. A connection relay can prevent a mechanical unit from being activated unless a specified digital signal is set.

Some tool changers support I/O signals that specify which gun is currently connected. Then a digital input signal from the tool changer is used by the connection relay.

If the tool changer does not support I/O signals, a similar behavior can be created with RAPID instructions. Set a digital output signal to 1 with the instruction `SetDO` each time the tool is connected, and set the signal to 0 when the tool is disconnected.

System parameters

This is a brief description of each parameter used to configure a connection relay. For more information, see *Technical reference manual - System parameters*.

The following parameters have to be set for the type *Mechanical Unit* in the topic *Motion*:

Parameter	Description
Use Connection Relay	The name of the relay to use. Corresponds to the name specified in the parameter <i>Name</i> in the type <i>Relay</i> .

The following parameters must be set for the type *Relay* in the topic *Motion*:

Parameter	Description
Name	Name of the relay. Used by the parameter <i>Use Connection Relay</i> in the type <i>Mechanical Unit</i> .
Input Signal	The name of the digital signal used to indicate if it should be possible to activate the mechanical unit.

Example of connection relay configuration

This is an example of how to configure connection relays for two gun tools. `gun1` can only be activated when signal `di1` is 1, and `gun2` can only be activated when `di2` is 1.

If the tool changer sets `di1` to 1 only when `gun1` is connected, and `di2` to 1 only when `gun2` is connected, there is no risk of activating the wrong gun.

The following parameter values are set for `gun1` and `gun2` in the type *Mechanical Unit*:

Name	Use Connection Relay
<code>gun1</code>	<code>gun1_relay</code>
<code>gun2</code>	<code>gun2_relay</code>

Continues on next page

10 Tool control options

10.1.4 Connection relay

Continued

The following parameter values are set for gun1 and gun2 in the type *Relay*:

Name	Input Signal
gun1_relay	di1
gun2_relay	di2

10.1.5 Tool change procedure

How to change tool

This is a description of how to change from gun1 to gun2.

Step	Action
1	Deactivate gun1 with the instruction: <code>DeactUnit gun1;</code>
2	Disconnect gun1 from the tool changer.
3	Connect gun2 to the tool changer.
4	Activate gun2 with the instruction: <code>ActUnit gun2;</code>
5	Optional but recommended: Calibrate gun2 with the instruction: <code>STCalib gun1 \ToolChg;</code> Note that this calibration requires option Servo Tool Control or Spot Servo.

10 Tool control options

10.1.6 Jogging servo tools with activation disabled

10.1.6 Jogging servo tools with activation disabled

Overview

Only one of the servo tools used by the tool changer may be activated at a time, the others are set to activation disabled. This is to make sure that the user is jogging the servo tool presently connected with right configuration.

What to do when Activation disabled appears

Follow these steps when you need to jog a servo tool but cannot activate the unit because activation is disabled.

Step	Action
1.	Make sure that the right servo tool is mounted on the tool changer. If the wrong tool is mounted, see Tool change procedure on page 305 .
2.	If no tool is activated, open the RAPID execution and activate the right tool.
3.	If the right tool is mounted on the tool changer, deactivate the wrong tool and activate the right tool from RAPID execution.

10.2 Tool Control [3109-1]

10.2.1 Overview

Purpose

Tool Control can be used to control a servo tool, for example in a spot weld application. *Tool Control* makes it possible to close the tool to a specific plate thickness and force, and maintain the force during the process until the tool is requested to be opened.

What is included

Tool Control gives you access to:

- RAPID instructions to open, close and calibrate servo tools
- RAPID instructions for tuning system parameter values
- RAPID functions for checking status of servo tools
- system parameters to configure servo tools

Basic approach

This is the general approach for using *Tool Control*.

- 1 Configure and calibrate the servo tool.
- 2 Perform a force calibration.
- 3 Create the RAPID program.

Prerequisites

A servo tool is an additional axis. Required hardware, such as drive module and measurement board, is specified in *Application manual - Additional axes*.

10 Tool control options

10.2.2 Servo tool movements

10.2.2 Servo tool movements

Closing and opening of a servo tool

The servo tool can be closed to a predefined thickness and tool force. When the tool reaches the programmed contact position, the movement is stopped and there is an immediate switch from position control mode to force control mode. In the force control mode a motor torque will be applied to achieve the desired tool force. The force remains constant until an opening is ordered. Opening of the tool will reduce the tool force to zero and move the tool arm back to the pre-close position.

Synchronous and asynchronous movements

Normally a servo tool axis is moved synchronous with the robot movements in such a way that both movements will be completed exactly at the same time. However the servo tool may be closed asynchronously (independent of current robot movement). The closing will immediately start to run the tool arm to the expected contact position (thickness). The closing movement will interrupt an on-going synchronous movement of the tool arm.

The tool opening may also take place while the robot is moving. But it is not possible if the robot movement includes a synchronized movement of the servo tool axis. A motion error, "tool opening could not synchronize with robot movement", will occur.

10.2.3 Tip management

About tip management

The tip management functionality will find and calibrate the contact position of the tool tips automatically. It will also update and monitor the total tip wear of the tool tips.

The tips can be calibrated using the RAPID instruction `STCalib` (see [Instructions on page 312](#)). Typically, two tool closings will be performed during a calibration.

Three different types of calibrations are supported: tip wear, tip change and tool change. All three will calibrate the contact position of the tips. The total tip wear will, however, be updated differently by these methods.

Tip wear calibration

As the tips are worn down, for example when spot welding, they need to be dressed. After the tip dressing, a tip wear calibration is required. The tool contact position is calibrated and the total tip wear of the tool is updated. The calibration movements are fast and the switch to force control mode will take place at the zero position.

This method must only be used to make small position adjustments (< 3 mm) caused by tip wear/tip dressing.



Tip

A variable in your RAPID program can keep track of the tip wear and inform you when the tips need to be replaced.

Tip change calibration

The tip change calibration is to be used after mounting a new pair of tips, for example when spot welding. The tool contact position is calibrated and the total tip wear of the tool is reset. The first calibration movement is slow in order to find the unknown contact position and switch to force control. The second calibration movement is fast. This calibration method will handle big position adjustments of the servo tool.

This calibration may be followed by a tool closing in order to squeeze the tips in place. A new tip change calibration is then done to update possible position differences after the tip squeeze.

Tool change calibration

The tool change calibration is to be used after reconnecting and activating a servo tool. The tool contact position is calibrated and the total tip wear of the tool remains unchanged. The first calibration movement is slow in order to find the unknown tip collision position and switch to force control. The second calibration movement is fast. This calibration method will handle big position adjustments of the tool.

The method should always be used after reconnecting a tool since the activation will restore the latest known position of the tool, and that position may be different from the actual tool position; the tool arm may have been moved when

Continues on next page

10 Tool control options

10.2.3 Tip management

Continued

disconnected. This calibration method will handle big position adjustments of the tool.



Tip

Tool change calibration is most commonly used together with the RobotWare option Servo Tool Change.

10.2.4 Supervision

Max and min stroke

An out of range supervision will stop the movement if the tool is reaching max stroke or if it is closed to contact with the tips (reaching min stroke). See *Upper Joint Bound* and *Lower Joint Bound* in [Arm on page 315](#).

Motion supervision

During the position control phase of the closing/opening, motion supervision is active for the servo tool to detect if the arm collides or gets stuck. A collision will cause a motion error and the motion will be stopped.

During the force control phase, the motion supervision will supervise the tool arm position not to exceed a certain distance from the expected contact position. See parameter *Max Force Control Position Error* in [Supervision Type on page 316](#).

Maximum torque

There is a maximum motor torque for the servo tool that never will be exceeded in order to protect the tool from damage. If the force is programmed out of range according to the tools force-torque table, the output force will be limited to this maximum allowed motor torque and a motion warning will be logged. See parameter *Max Force Control Motor Torque* in [SG Process on page 313](#).

Speed limit

During the force control phase there is a speed limitation. The speed limitation will give a controlled behavior of the tool even if the force control starts before the tool is completely closed. See *Speed limit 1- 6* in [Force Master Control on page 314](#).

10 Tool control options

10.2.5 RAPID components

10.2.5 RAPID components

About the RAPID components

This is an overview of all instructions, functions, and data types in *Tool Control*.
For more information, see *Technical reference manual - RAPID Instructions, Functions and Data types*.

Instructions

Instruction	Description
STClose	Close the servo tool with a predefined force and thickness.
STOpen	Open the servo tool.
STCalib	Calibrate the servo tool. An argument determines which type of calibration will be performed: <ul style="list-style-type: none">• <code>\ToolChg</code> for tool change calibration• <code>\TipChg</code> for tip change calibration• <code>\TipWear</code> for tip wear calibration
STTune	Tune motion parameters for the servo tool. A temporary value can be set for a parameter specified in the instruction.
STTuneReset	Reset tuned motion parameters for the servo tool. Cancel the effect of all <code>STTune</code> instructions.

Functions

Function	Description
STIsClosed	Test if the servo tool is closed.
STIsOpen	Test if the servo tool is open.
STIsCalib	Tests if a servo tool is calibrated.
STCalcTorque	Calculate the motor torque for a servo tool.
STCalcForce	Calculate the force for a servo tool.
STIsServoTool	Tests if a mechanical unit is a servo tool.
STIsIndGun	Tests if servo tool is in independent mode.

Data types

Tool Control includes no RAPID data types.

10.2.6 System parameters

About the system parameters

When using a servo tool, a motion parameter file for the tool is normally installed on the controller. A servo tool is a specific variant of an additional axis and the description of how to configure the servo tool is found in *Application manual - Additional axes*.

In this section, the parameters used in combination with *Tool Control* is briefly described. For more information, see the respective parameter in *Technical reference manual - System parameters*.

SG Process

These parameters belong to the type *SG Process* in the topic *Motion*.

SG Process is used to configure the behavior of a servo gun (or other servo tool).

Parameter	Description
<i>Close Time Adjust</i>	Adjustment of the ordered minimum close time of the gun.
<i>Close Position Adjust</i>	Adjustment of the ordered position (plate thickness) where force control should start, when closing the gun.
<i>Force Ready Delay</i>	Delays the close ready event after achieving the ordered force.
<i>Max Force Control Motor Torque</i>	Max allowed motor torque for force control. Commanded force will be reduced, if the required motor torque is higher than this value.
<i>Post-synchronization Time</i>	Anticipation of the open ready event. This can be used to synchronize the gun opening with the next robot movement.
<i>Calibration Mode</i>	Defines the number of times the servo gun closes during a tip wear calibration.
<i>Calibration Force Low</i>	The minimum tip force used during a tip wear calibration.
<i>Calibration Force High</i>	The maximum tip force used during a tip wear calibration.
<i>Calibration Time</i>	The time that the servo gun waits in closed position during calibration.
<i>Number of Stored Forces</i>	Defines the number of points in the force-torque relation specified in <i>Tip Force 1 - 10</i> and <i>Motor Torque 1 - 10</i> .
<i>Tip Force 1 - 10</i>	<i>Tip Force 1</i> defines the tip force that corresponds to the motor torque in <i>Motor Torque 1</i> . <i>Tip Force 2</i> corresponds to <i>Motor Torque 2</i> , etc.
<i>Motor Torque 1- 10</i>	<i>Motor Torque 1</i> defines the motor torque that corresponds to the tip force in <i>Tip Force 1</i> . <i>Motor Torque 2</i> corresponds to <i>Tip Force 2</i> , etc.
<i>Squeeze Position 1 - 10</i>	Defines the joint position at each force level in the force calibration table.
<i>Soft Stop Timeout</i>	Defines how long the force will be maintained if a soft stop occurs during constant force.

Continues on next page

10 Tool control options

10.2.6 System parameters

Continued

Force Master

These parameters belong to the type *Force Master* in the topic *Motion*.

Force Master is used to define how a servo tool, typically a servo gun, behaves during force control. The parameters only affect the servo tool when it is in force control mode.

Parameter	Description
<i>References Bandwidth</i>	The frequency limit for the low pass filter for reference values.
<i>Use ramp time</i>	Determines if the ramping of the tip force should use a constant time or a constant gradient.
<i>Ramp when Increase Force</i>	Determines how fast force is built up while closing the tool when <i>Use ramp time</i> is set to No.
<i>Ramp time</i>	Determines how fast force is built up while closing the tool when <i>Use ramp time</i> is set to Yes.
<i>Collision LP Bandwidth</i>	Frequency limit for the low pass filter used for tip wear calibration.
<i>Collision Alarm Torque</i>	Determines how hard the tool tips will be pressed together during the first gun closing of new tips calibrations and tool change calibrations.
<i>Collision Speed</i>	Determines the servo gun speed during the first gun closing of new tips calibrations and tool change calibrations.
<i>Collision Delta Position</i>	Defines the distance the servo tool has gone beyond the contact position when the motor torque has reached the value specified in <i>Collision Alarm Torque</i> .
<i>Max pos err. closing</i>	Determines how close to the ordered plate thickness the tool tips must be before the force control starts.
<i>Delay ramp</i>	Delays the starting of torque ramp when force control is started.
<i>Ramp to real contact</i>	Determines if the feedback position should be used instead of reference position when deciding the contact position.

Force Master Control

These parameters belong to the type *Force Master Control* in the topic *Motion*.

Force Master Control is used to set the speed limit and speed loop gain as functions of the torque.

Parameter	Description
<i>No. of speed limits</i>	The number of points used to define speed limit and speed loop gain as functions of the torque. Up to 6 points can be defined.
<i>torque 1 - torque 6</i>	The torque levels, corresponding to the ordered tip force, for which the speed limit and speed loop gain values are defined.
<i>Speed Limit 1 - 6</i>	<i>Speed Limit 1</i> to <i>Speed Limit 6</i> are used to define the maximum speed depending on the ordered tip force.
<i>Kv 1 - 6</i>	<i>Kv 1</i> to <i>Kv 6</i> are used to define the speed loop gain for reducing the speed when the speed limit is exceeded.

Continues on next page

Arm

These parameters belong to the type *Arm* in the topic *Motion*.

The type *Arm* defines the characteristics of an arm.

Parameter	Description
<i>Upper Joint Bound</i>	Defines the upper limit of the working area for the joint.
<i>Lower Joint Bound</i>	Defines the lower limit of the working area for the joint.

Acceleration Data

These parameters belong to the type *Acceleration Data* in the topic *Motion*.

Acceleration Data is used to specify some acceleration characteristics for axes without any dynamic model.

Parameter	Description
<i>Nominal Acceleration</i>	Worst case motor acceleration.
<i>Nominal Deceleration</i>	Worst case motor deceleration.
<i>Acceleration Derivate Ratio</i>	Indicates how fast the acceleration can be increased.
<i>Deceleration Derivate Ratio</i>	Indicates how fast the deceleration can be increased.

Motor Type

These parameters belong to the type *Motor Type* in the topic *Motion*.

Motor Type is used to describe characteristics for a motor.

Parameter	Description
<i>Pole Pairs</i>	Defines the number of pole pairs for the motor.
<i>Inertia</i>	The inertia of the motor, including the resolver but excluding the brake.
<i>Stall Torque</i>	The continuous stall torque, i.e. the torque the motor can produce at no speed and during an infinite time.
<i>ke Phase to Phase</i>	Nominal voltage constant. The induced voltage (phase to phase) that corresponds to the speed 1 rad/s.
<i>Max Current</i>	Max current without irreversible magnetization.
<i>Phase Resistance</i>	Nominal winding resistance per phase at 20 degrees Celsius.
<i>Phase Inductance</i>	Nominal winding inductance per phase at zero current.

Motor Calibration

These parameters belong to the type *Motor Calibration* in the topic *Motion*.

Motor Calibration is used to calibrate a motor.

Parameter	Description
<i>Commutator Offset</i>	Defines the position of the motor (resolver) when the rotor is in the electrical zero position relative to the stator.
<i>Calibration Offset</i>	Defines the position of the motor (resolver) when it is in the calibration position.

Continues on next page

10 Tool control options

10.2.6 System parameters

Continued

Stress Duty Cycle

These parameters belong to the type *Stress Duty Cycle* in the topic *Motion*.

Stress Duty Cycle is used for protecting axes, gearboxes, etc.

Parameter	Description
<i>Speed Absolute Max</i>	The absolute highest motor speed to be used.
<i>Torque Absolute Max</i>	The absolute highest motor torque to be used.

Supervision Type

These parameters belong to the type *Supervision Type* in the topic *Motion*.

Supervision Type is used for continuous supervision of position, speed and torque.

Parameter	Description
<i>Max Force Control Position Error</i>	When a servo gun is in force control mode it is not allowed to move more than the distance specified in <i>Max Force Control Position Error</i> . This supervision will protect the tool if, for instance, one tip is lost.
<i>Max Force Control Speed Limit</i>	Speed error factor during force control. If the speed limits, defined in the type <i>Force Master Control</i> , multiplied with <i>Max Force Control Speed Limit</i> is exceeded, all movement is stopped.

Transmission

These parameters belong to the type *Transmission* in the topic *Motion*.

Transmission is used to define the transmission gear ratio between a motor and its axis.

Parameter	Description
<i>Rotating Move</i>	Defines if the axis is rotating or linear.
<i>Transmission Gear Ratio</i>	Defines the transmission gear ratio between motor and joint.

Lag Control Master 0

These parameters belong to the type *Lag Control Master 0* in the topic *Motion*.

Lag Control Master 0 is used for regulation of axes without any dynamic model.

Parameter	Description
<i>FFW Mode</i>	Defines if the position regulation should use feed forward of speed and torque values.
<i>Kp, Gain Position Loop</i>	Proportional gain in the position regulation loop.
<i>Kv, Gain Speed Loop</i>	Proportional gain in the speed regulation loop.
<i>Ti Integration Time Speed Loop</i>	Integration time in the speed regulation loop.

Continues on next page

Uncalibrated Control Master 0

These parameters belong to the type *Uncalibrated Control Master 0* in the topic *Motion*.

Uncalibrated Control Master 0 is used to regulate uncalibrated axes.

Parameter	Description
<i>Kp, Gain Position Loop</i>	Proportional gain in the position regulation loop.
<i>Kv, Gain Speed Loop</i>	Proportional gain in the speed regulation loop.
<i>Ti Integration Time Speed Loop</i>	Integration time in the speed regulation loop.
<i>Speed Max Uncalibrated</i>	The maximum allowed speed for an uncalibrated axis.
<i>Acceleration Max Uncalibrated</i>	The maximum allowed acceleration for an uncalibrated axis.
<i>Deceleration Max Uncalibrated</i>	The maximum allowed deceleration for an uncalibrated axis.

10 Tool control options

10.2.7 Commissioning and service

10.2.7 Commissioning and service

Commissioning the servo tool

For a new servo tool, follow these steps for installing and commissioning:

Step	Action
1	Install the servo tool according to the description in <i>Application manual - Additional axes</i> .
2	Load a <i>.cfg</i> file with the servo tool configuration. For detailed description on how to do this, see <i>Operating manual - RobotStudio</i> . If you do not have any <i>.cfg</i> file for the servo tool, you can load a template file and configure the system parameters with the values of your servo tool. Template files are found in the RobotWare distribution, see Template file locations on page 318 .
3	Use the RAPID instruction <code>STTune</code> and iterate to find the optimal parameter values. Once found, these optimal values should be written to the system parameters to be permanent.
4	Fine calibrate the servo tool, see Fine calibration on page 320 .
5	Unless force calibration was included in a loaded <i>.cfg</i> file, perform a force calibration.

Template file locations

The template files can be obtained from the PC or the IRC5 controller.

- In the RobotWare installation folder in RobotStudio: `...\RobotPackages\RobotWare_RPK_<version>\utility\AdditionalAxis\`
- On the IRC5 Controller:
`<SystemName>\PRODUCTS\<RobotWare_xx.xx.xxxx>\utility\AdditionalAxis\`



Note

Navigate to the RobotWare installation folder from the RobotStudio **Add-Ins** tab, by right-clicking on the installed RobotWare version in the **Add-Ins** browser and selecting **Open Package Folder**.

Disconnect/reconnect a servo tool

If the servo tool is deactivated, using the `DeactUnit` instruction, it may be disconnected and removed. The tool position at deactivation will be restored when the tool is connected and reactivated. Make a tool change calibration to make sure the tip position is OK.

The whole process of changing a tool can be performed by a RAPID program if you use the RobotWare option Servo Tool Change and the instruction `STCalib`.

Recover from accidental disconnection

If the motor cables are disconnected by accident when the servo tool is active, the system will go into system failure state. After restart of the system the servo tool must be deactivated in order to jog the robot to a service position.

Deactivation may be performed from the **Jogging** window. Tap on **Activate...**, select the servo tool and tap on **Deactivate**.

Continues on next page

After service / repair the revolution counter must be updated since the position has been lost, see [Update revolution counter on page 320](#).

10 Tool control options

10.2.8 Mechanical unit calibrations

10.2.8 Mechanical unit calibrations

Fine calibration

Fine calibration must be performed when installing a new servo tool, or if the servo tool axis is in state 'Not Calibrated'.

For this, it is recommended to create a service routine using the following instructions:

```
STCalib "ToolName" \TipChg;  
STCalib "ToolName" \TipWear;
```

Update revolution counter

An update of the revolution counter must be performed if the position of the axis is lost. If this happens, this is indicated by the calibration state 'Rev. Counter not updated'.

For this, it is recommended to use the same service routine as for the fine calibration.

10.2.9 RAPID code example

How to use the code package

The normal programming technique for *Tool Control* is to customize shell routines based on the example code below. These shell routines are then called from your program.

Using shell routines

This example shows a main routine in combination with a customized routine (`rMoveSpot`) that uses the standard servo tool instructions. The external process (for example a weld timer) is indicated with the routine `rWeld`.

```

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 (robtarget 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;
  STClose Gun, Thickness;
  rWeld;
  STOpen 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

```

This page is intentionally left blank

Index

3

3rd party software, 14

A

Absolute Accuracy, 177
 Absolute Accuracy calibration, 187
 Absolute Accuracy compensation, 185
 Absolute Accuracy verification, 188
 Acceleration Data, 315
 Acceleration Derivate Ratio, 315
 Acceleration Max Uncalibrated, 317
 accidental disconnection, 318
 acknowledge messages, 171
 activate Absolute Accuracy, 180
 activate supervision, 243
 activation disabled, 306
 actor signals, 153–154
 additional axes, 307
 additional axis, 116
 Advanced RAPID, 19
 Advanced Shape Tuning, 201
 AliasIO, 26–27
 alignment, 191
 analog signal, 50
 Analog Signal Interrupt, 50
 AND, 154
 ArgName, 48
 argument name, 48
 Arm, 315
 arm replacement, 181
 asynchronous movements, 308
 Auto acknowledge input, 12–13, 297
 automatic friction tuning, 202
 axis, 251
 axis reset, 251

B

binary communication, 135
 binary data, 171
 birth certificate, Absolute Accuracy, 189
 BitAnd, 21
 BitCheck, 21
 BitClear, 21
 bit functionality, 20
 BitLSh, 21
 BitNeg, 21
 BitOr, 21
 BitRSh, 21
 BitSet, 21
 BitXOr, 21
 BookErrNo, 43
 byte, 21
 ByteToStr, 21

C

calibrate follower axis, 123
 calibrate tool, 195
 calibration data, 180
 Calibration Force High, 313
 Calibration Force Low, 313
 Calibration Mode, 313
 Calibration Offset, 315
 calibration process, 187
 Calibration Time, 313

calibration tools, 179
 CalibWare, 179
 cell alignment, 191
 certificate, Absolute Accuracy, 189
 change calibration data, 180
 character based communication, 135
 Check unresolved references, Task type, 267
 CirPathMode, 221
 ClearRawBytes, 140
 Close, 136
 CloseDir, 144
 Close position adjust, 313
 Close time adjust, 313
 code example, 321
 collision, 234
 Collision Alarm Torque, 314
 Collision Avoidance, 245
 Collision Delta Position, 314
 collision detection
 YuMi robots, 232
 Collision Detection Memory, 237
 Collision Error Handler, 238
 Collision LP Bandwidth, 314
 Collision Speed, 314
 commissioning, 318
 common data, 271
 communication, 134
 Commutator Offset, 315
 compensation, 185
 compensation parameters, 177, 190
 compliance errors, 184
 configuration
 Absolute Accuracy, 180
 configuration functionality, 29
 configure Collision Detection, 241
 connection relay, 303
 coordinate systems, 191
 CopyFile, 144
 CopyRawBytes, 140
 Corr argument, 295
 CorrClear, 294
 CorrCon, 294
 corrdescr, 294
 CorrDiscon, 294
 correction generator, 292
 CorrRead, 294
 CorrWrite, 294
 cross connections, 153
 cut plane, 219
 cut shape, 224
 Cyclic bool, 104
 Cyclic bool settings, 110
 Cyclic bool system parameters, 110

D

data, 159
 datapos, 24
 data search functionality, 23
 data types
 Multitasking, 269
 data variable example
 Electronically Linked Motors, 132
 data variables
 Electronically Linked Motors, 130
 Deactivate PTC superv. at disconnect, 302
 deactivate supervision, 243
 Deceleration Derivate Ratio, 315

- Deceleration Max Uncalibrated, 317
- declarations, 271
- deflection, 185
- Delay ramp, 314
- digital I/O signals, 153
- dir, 144
- directory management, 143
- discarded message, 161
- Disconnect at Deactivate, 302
- disconnection, 318
- dispatcher, 276
- displacement, 131

E

- Electronically Linked Motors, 116
- errdomain, 40
- error interrupts, 39
- error sources in accuracy, 184
- ErrRaise, 40
- errtype, 40
- Ethernet, 283, 285
- event messages, 42
- event number, 42
- Event Preset Time, 150
- external axes, 233
- external axis, 251

F

- fake target, 185
- false triggering, 244
- FFW Mode, 316
- Fieldbus Command Interface, 112
- FIFO, 160
- file communication, 134
- file management, 143
- FileSize, 144
- file structures, 143
- fine calibration, 320
- fixed position events, 147
- fixture alignment, 192
- FlexPendant, 278
- follower, 116
- Follower to Joint, 118
- Force Master, 314
- Force Master Control, 314
- Force Ready Delay, 313
- frame relationships, 194
- frames, 191
- FricIdEvaluate, 208
- FricIdInit, 208
- FricIdSetFricLevels, 208
- friction compensation, 201
- Friction FFW Level, 206
- Friction FFW On, 206
- Friction FFW Ramp, 206
- friction level tuning, 202
- FSSize, 144
- functions
 - Advanced RAPID, 48
 - Multitasking, 269

G

- General RAPID, 238
- GetDataVal, 24
- GetMaxNumberOfCyclicBool, 111
- GetNextCyclicBool, 111
- GetNextSym, 24

- GetNumberOfCyclicBool, 111
- GetTrapData, 40
- group I/O signals, 153

I

- IError, 40
- IndAMove, 254
- IndCMove, 254
- Ind collision stop without brake, 238
- IndDMove, 254
- Independent Axes, 251
- independent joint, 233
- Independent Joint, 253
- Independent Lower Joint Bound, 253
- independent movement, 251
- Independent Upper Joint Bound, 253
- IndInpos, 254
- IndReset, 254
- IndRMove, 254
- IndSpeed, 254
- Inertia, 315
- Input Signal, 303
- installation, 318
- instructions
 - Advanced RAPID, 48
 - Multitasking, 269
- interrupt, 50, 160, 272
- interrupt functionality, 39
- iodev, 136
- IPers, 40
- IRMQMessage, 164
- IsCyclicBool, 111
- IsFile, 144
- ISignalAI, 51
- ISignalAO, 51
- IsStopStateEvent, 48

J

- Jog Collision Detection, 237, 241
- Jog Collision Detection Level, 237
- Jog Collision Detection Level, 241
- Joint, 118
- joint zones, 225

K

- ke Phase to Phase, 315
- kinematic errors, 184
- Kp, Gain Position Loop, 316–317
- Kv 1 - 6, 314
- Kv, Gain Speed Loop, 316
- Kv, Gain Speed Loop, 317

L

- l_f_axis_name, 130
- l_f_axis_no, 130
- l_f_mecunt_n, 130
- l_m_axis_no, 130
- l_m_mecunt_n, 130
- Lag Control Master 0, 316
- licenses, 14
- Linked M Process, 118
- load calibration data, 180
- Load Identification, 179
- Lock Joint in Ipol, 118
- logical AND, 155
- Logical Axis, 302
- Logical Cross Connections, 153

logical operations, 153
 logical OR, 155
 loss of accuracy, 183
 lost message, 161
 lost queue, 161
 Lower Joint Bound, 315

M

Main entry, Task type, 267
 maintenance, 181
 MakeDir, 144
 manipulator replacement, 182
 Manipulator Supervision, 237
 Manipulator Supervision Level, 237
 manual friction tuning, 204
 master, 116
 Master Follower kp, 119
 Max Current, 315
 Max Follower Offset, 118
 Max Force Control Motor Torque, 313
 Max Force Control Position Error, 316
 Max Force Control Speed Limit, 316
 Max Offset Speed, 118
 Max pos err. closing, 314
 measurement system, 254
 mechanical unit, 279
 merge of messages, 171
 Motion Planner, 237
 Motion Process Mode, 209
 MotionSup, 239, 243
 Motion Supervision, 237
 Motion Supervision Max Level, 237
 Motion System, 238
 MotionTask, Task type, 268
 Motor Calibration, 315
 motor replacement, 181
 Motor Torque 1- 10, 313
 Motor Type, 315
 MotSupOn, 240
 MotSupTrigg, 240
 MoveCSync, 149
 MoveJSync, 149
 MoveLSync, 149
 Multitasking, 265

N

NFS Client, 285
 No. of speed limits, 314
 Nominal Acceleration, 315
 Nominal Deceleration, 315
 non printable characters, 171
 NORMAL, 267
 NoSafety, 267
 NOT, 155
 Not Calibrated, 320
 Number of Stored Forces, 313

O

offset_ratio, 130
 Offset Adjust Delay Time, 118
 Offset Speed Ratio, 118
 Open, 136
 OpenDir, 144
 open source software, OSS, 14
 OR, 154

P

PackDNHeader, 113
 PackRawBytes, 140
 parameters
 accuracy compensation, 190
 path, 33
 Path Collision Detection, 237, 241
 Path Collision Detection Level, 237, 241
 path correction, 292
 path offset, 292
 pathrecid, 258
 PathRecMoveBwd, 258
 PathRecMoveFwd, 258
 path recorder, 261
 Path Recovery, 257
 PathRecStart, 258
 PathRecStop, 258
 PathRecValidBwd, 258
 PathRecValidFwd, 258
 PC SDK client, 159
 persistent variables, 270
 PFRestart, 33
 Phase Inductance, 315
 Phase Resistance, 315
 pitch, 184
 Pole Pairs, 315
 polling, 272
 position event, 147
 Post-synchronization Time, 313
 power failure functionality, 33
 Process, 118
 process support functionality, 35
 program pointer, 48
 proportional signal, 36

Q

queue handling, 160
 queue name, 160

R

r1_calib, 180
 Ramp time, 314
 Ramp Time, 119
 Ramp to real contact, 314
 Ramp when Increase Force, 314
 RAPID, 17
 RAPID components
 Advanced RAPID, 48
 Multitasking, 269
 RAPID Message Queue, 158
 RAPID support functionality, 47
 rawbytes, 140
 RawBytesLen, 140
 raw data, 139
 ReadAnyBin, 136
 ReadBin, 136
 ReadCfgData, 30
 ReadDir, 144
 ReadErrData, 40
 ReadNum, 136
 ReadRawBytes, 140
 ReadStr, 136
 ReadStrBin, 136
 reconnect a servo tool, 318
 record, 159
 recorded path, 261
 recover path, 257

References Bandwidth, 314
relay, 303
RemoveAllCyclicBool, 111
RemoveCyclicBool, 111
RemoveDir, 144
RemoveFile, 144
RenameFile, 144
replacements, 181
reset, 254
reset axis, 251
reset follower axis, 125
resolver offset calibration, 187
restartdata, 36
RestoPath, 258
resultant signal, 153–154
resume signals, 37
Rev. Counter not updated, 320
reversed movement, 234
Rewind, 136
RMQEmptyQueue, 164
RMQFindSlot, 164
RMQGetMessage, 164
RMQGetMsgData, 164
RMQGetMsgHeader, 164
RMQGetSlotName, 164
rmqheader, 164
RMQ Max Message Size, 163
RMQ Max No Of Messages, 163
rmqmessage, 164
RMQ Mode, 163
RMQReadWait, 164
RMQSendMessage, 164
RMQSendWait, 164
rmqslot, 164
RMQ Type, 163
robot alignment, 193
roll, 184
Rotating Move, 316
routine call, 276

S

SafeMove Assistant, 248
SEMISTATIC, 267
sensor, 292
service, 318
service routines
 Electronically Linked Motors, 121
Servo Tool Change, 299
SetAllDataVal, 24
SetDataSearch, 24
SetDataVal, 24
SetSysData, 48
set up Collision Detection, 241
SetupCyclicBool, 111
SG Process, 313
shapedata, 227
shared resources, 278
signal, 272, 276
SocketAccept, 172
SocketBind, 172
SocketClose, 172
SocketConnect, 172
SocketCreate, 172
socketdev, 172
SocketGetStatus, 173
SocketListen, 172
Socket Messaging, 169

SocketReceive, 172
SocketSend, 172
socketstatus, 172
soft servo, 233
Soft Stop Timeout, 313
software licenses, 14
speed, 235
speed_ratio, 130
Speed Absolute Max, 316
Speed Limit 1 - 6, 314
Speed Max Uncalibrated, 317
Squeeze Position 1 -10, 313
Stall Torque, 315
STATIC, 267
stationary world zone, 227
STCalcForce, 312
STCalcTorque, 312
STCalib, 312
STClose, 312
StepBwdPath, 36
STIsCalib, 312
STIsClosed, 312
STIsIndGun, 312
STIsOpen, 312
STIsServoTool, 312
STOpen, 312
StorePath, 258
Stress Duty Cycle, 316
string termination, 171
StrToByte, 21
STTune, 312
STTuneReset, 312
supervision level, 237, 239, 243
Supervision Type, 316
synchronizing tasks, 274
synchronous movements, 308
syncident, 274
syncident, data type, 269
SyncMoveResume, 258
SyncMoveSuspend, 258
SysFail, 267
SysHalt, 267
SysStop, 267
system parameters
 configuration functionality, 29
 Multitasking, 267
system resources, 278

T

Task, Task type, 267
Task, type, 267
taskid, 269, 280
taskid, data type, 269
Task in foreground, Task type, 267
TaskRunMec, 279
TaskRunMec, function, 269
TaskRunRob, 279
TaskRunRob, function, 269
tasks, 265, 274
 data type, 269
tasks, data type, 269
temporary world zone, 227
TestAndSet, 278
TestAndSet, function, 269
TextGet, 43
TextTabFreeToUse, 43
TextTabGet, 43

TextTabInstall, 43
 text table file, 42
 Ti Integration Time Speed Loop, 316–317
 tip change calibration, 309
 Tip Force 1 - 10, 313
 tip wear calibration, 309
 tool calibration, 195
 tool change calibration, 309
 tools, 179
 torque, 235
 torque 1 - torque 6, 314
 Torque Absolute Max, 316
 torque follower, 126
 track motion, 233
 Transmission, 316
 Transmission Gear High, 253
 Transmission Gear Low, 253
 Transmission Gear Ratio, 316
 trapdata, 40
 trap routine, 160
 TriggC, 149
 TriggCheckIO, 149
 triggdata, 148
 TriggEquip, 148
 triggering, 244
 TriggInt, 148
 TriggIO, 148
 triggios, 148
 triggiosdnum, 148
 TriggJ, 149
 TriggL, 149
 TriggLIOs, 149
 TriggRampAO, 149
 TriggSpeed, 36
 TriggStopProc, 36
 triggstrgo, 148
 TrustLevel, Task type, 267
 TUNE_FRIC_LEV, 204
 TUNE_FRIC_RAMP, 204
 TuneServo, 204
 tuning, 243
 tuning, automatic, 202
 tuning, manual, 204
 Type, Task type, 267

U

uncalib, 180
 Uncalibrated Control Master 0, 317

Unicode, 17
 UnpackRawBytes, 140
 unsynchronize, 123
 Update revolution counter, 320
 Upper Joint Bound, 315
 Use Connection Relay, 303
 Use Linked Motor Process, 118
 Use Process, 118
 Use ramp time, 314
 user message functionality, 42
 Use Robot Calibration, 180

V

verification, 188

W

waiting for tasks, 274
 WaitSyncTask, 274
 WaitSyncTask, instruction, 269
 WaitUntil, 272
 WarmStart, 30
 world zones, 225
 Wrist Move, 217
 wrist replacement, 181
 Write, 136
 WriteAnyBin, 136
 WriteBin, 136
 WriteCfgData, 30
 WriteRawBytes, 140
 WriteStrBin, 136
 WZBoxDef, 227
 WZCylDef, 227
 WZDisable, 228
 WZDSet, 228
 WZEnable, 228
 WZFree, 228
 WZHomeJointDef, 228
 WZLimJointDef, 228
 WZLimSup, 228
 WZSphDef, 227
 wzstationary, 227
 wztemporary, 227

Y

yaw, 184

Z

zones, 225



ABB AB

Robotics & Discrete Automation

S-721 68 VÄSTERÅS, Sweden

Telephone +46 10-732 50 00

ABB AS

Robotics & Discrete Automation

Nordlysvegen 7, N-4340 BRYNE, Norway

Box 265, N-4349 BRYNE, Norway

Telephone: +47 22 87 2000

ABB Engineering (Shanghai) Ltd.

Robotics & Discrete Automation

No. 4528 Kangxin Highway

PuDong New District

SHANGHAI 201319, China

Telephone: +86 21 6105 6666

ABB Inc.

Robotics & Discrete Automation

1250 Brown Road

Auburn Hills, MI 48326

USA

Telephone: +1 248 391 9000

abb.com/robotics