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.
# Table of contents

Overview of this manual ........................................................................................................ 11
Open source and 3rd party components in RobotWare ...................................................... 13

1 Introduction to RobotWare ................................................................................................. 15

2 RobotWare-OS ...................................................................................................................... 17
   2.1 Advanced RAPID ........................................................................................................... 17
      2.1.1 Introduction to Advanced RAPID ........................................................................ 17
      2.1.2 Bit functionality .................................................................................................... 18
         2.1.2.1 Overview ..................................................................................................... 18
         2.1.2.2 RAPID components ..................................................................................... 19
         2.1.2.3 Bit functionality example .............................................................................. 20
      2.1.3 Data search functionality ......................................................................................... 21
         2.1.3.1 Overview ..................................................................................................... 21
         2.1.3.2 RAPID components ..................................................................................... 22
         2.1.3.3 Data search functionality examples .............................................................. 23
      2.1.4 Alias I/O signals ..................................................................................................... 24
         2.1.4.1 Overview ..................................................................................................... 24
         2.1.4.2 RAPID components ..................................................................................... 25
         2.1.4.3 Alias I/O functionality example ..................................................................... 26
      2.1.5 Configuration functionality ...................................................................................... 27
         2.1.5.1 Overview ..................................................................................................... 27
         2.1.5.2 RAPID components ..................................................................................... 28
         2.1.5.3 Configuration functionality example .............................................................. 29
      2.1.6 Power failure functionality ....................................................................................... 30
         2.1.6.1 Overview ..................................................................................................... 30
         2.1.6.2 RAPID components and system parameters ................................................ 31
         2.1.6.3 Power failure functionality example .............................................................. 32
      2.1.7 Process support functionality ................................................................................. 33
         2.1.7.1 Overview ..................................................................................................... 33
         2.1.7.2 RAPID components ..................................................................................... 34
         2.1.7.3 Process support functionality examples ........................................................ 35
      2.1.8 Interrupt functionality ............................................................................................. 37
         2.1.8.1 Overview ..................................................................................................... 37
         2.1.8.2 RAPID components ..................................................................................... 38
         2.1.8.3 Interrupt functionality examples ................................................................. 39
      2.1.9 User message functionality ...................................................................................... 40
         2.1.9.1 Overview ..................................................................................................... 40
         2.1.9.2 RAPID components ..................................................................................... 41
         2.1.9.3 User message functionality examples ............................................................. 42
         2.1.9.4 Text table files ............................................................................................. 44
      2.1.10 RAPID support functionality ................................................................................. 45
         2.1.10.1 Overview ................................................................................................... 45
         2.1.10.2 RAPID components ................................................................................... 46
         2.1.10.3 RAPID support functionality examples ......................................................... 47
   2.2 Analog Signal Interrupt ................................................................................................. 48
      2.2.1 Introduction to Analog Signal Interrupt ............................................................... 48
      2.2.2 RAPID components ............................................................................................. 49
      2.2.3 Code example ...................................................................................................... 50
   2.3 Connected Services ......................................................................................................... 51
      2.3.1 Overview .............................................................................................................. 51
      2.3.2 Connected Services connectivity ........................................................................ 53
      2.3.3 Connected Services registration ........................................................................ 54
      2.3.4 Summary of Connected Services paths in FlexPendant .................................... 56
      2.3.5 Summary of Connected Services paths in RobotStudio .................................... 57
      2.3.6 Configuration - system parameters .................................................................... 58
### Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.7 Configuring Connected Services using FlexPendant</td>
<td>62</td>
</tr>
<tr>
<td>2.3.7.1 Introduction</td>
<td>62</td>
</tr>
<tr>
<td>2.3.7.2 Enable or disable Connected Services using FlexPendant</td>
<td>63</td>
</tr>
<tr>
<td>2.3.7.3 Configure Connected Services based on connection type using</td>
<td>64</td>
</tr>
<tr>
<td>FlexPendant</td>
<td></td>
</tr>
<tr>
<td>2.3.7.4 Configuration of public network using FlexPendant</td>
<td>67</td>
</tr>
<tr>
<td>2.3.7.5 Configure internet connection with proxy using FlexPendant</td>
<td>68</td>
</tr>
<tr>
<td>2.3.8 Configuring Connected Services using RobotStudio</td>
<td>69</td>
</tr>
<tr>
<td>2.3.8.1 Introduction</td>
<td>69</td>
</tr>
<tr>
<td>2.3.8.2 Enable or disable connected services using RobotStudio</td>
<td>70</td>
</tr>
<tr>
<td>2.3.8.3 Configure connected services based on connection type using</td>
<td>71</td>
</tr>
<tr>
<td>RobotStudio</td>
<td></td>
</tr>
<tr>
<td>2.3.8.4 Configuration of public network using RobotStudio</td>
<td>74</td>
</tr>
<tr>
<td>2.3.8.5 Configure internet connection with proxy using RobotStudio</td>
<td>75</td>
</tr>
<tr>
<td>2.3.9 Connected Services information</td>
<td>76</td>
</tr>
<tr>
<td>2.3.10 Troubleshooting</td>
<td>88</td>
</tr>
<tr>
<td>2.3.10.1 Server connectivity troubleshooting</td>
<td>88</td>
</tr>
<tr>
<td>2.3.10.2 3G / Wi-Fi Connectivity troubleshooting</td>
<td>90</td>
</tr>
<tr>
<td>2.3.10.3 How to get Connected Services Embedded logs from the controller</td>
<td>91</td>
</tr>
<tr>
<td>2.3.10.4 Connected Services Embedded troubleshooting logs</td>
<td>92</td>
</tr>
<tr>
<td>2.3.11 Network topology scenarios</td>
<td>94</td>
</tr>
<tr>
<td>2.4 Cyclic bool</td>
<td>104</td>
</tr>
<tr>
<td>2.4.1 Cyclically evaluated logical conditions</td>
<td>104</td>
</tr>
<tr>
<td>2.4.2 Cyclic bool examples</td>
<td>107</td>
</tr>
<tr>
<td>2.4.3 System parameters</td>
<td>110</td>
</tr>
<tr>
<td>2.4.4 RAPID components</td>
<td>111</td>
</tr>
<tr>
<td>2.5 Device Command Interface</td>
<td>112</td>
</tr>
<tr>
<td>2.5.1 Introduction to Device Command Interface</td>
<td>112</td>
</tr>
<tr>
<td>2.5.2 RAPID components and system parameters</td>
<td>113</td>
</tr>
<tr>
<td>2.5.3 Code example</td>
<td>114</td>
</tr>
<tr>
<td>2.6 File and I/O device handling</td>
<td>116</td>
</tr>
<tr>
<td>2.6.1 Introduction to file and I/O device handling</td>
<td>116</td>
</tr>
<tr>
<td>2.6.2 Binary and character based communication</td>
<td>117</td>
</tr>
<tr>
<td>2.6.2.1 Overview</td>
<td>117</td>
</tr>
<tr>
<td>2.6.2.2 RAPID components</td>
<td>118</td>
</tr>
<tr>
<td>2.6.2.3 Code examples</td>
<td>119</td>
</tr>
<tr>
<td>2.6.3 Raw data communication</td>
<td>121</td>
</tr>
<tr>
<td>2.6.3.1 Overview</td>
<td>121</td>
</tr>
<tr>
<td>2.6.3.2 RAPID components</td>
<td>122</td>
</tr>
<tr>
<td>2.6.3.3 Code examples</td>
<td>123</td>
</tr>
<tr>
<td>2.6.4 File and directory management</td>
<td>125</td>
</tr>
<tr>
<td>2.6.4.1 Overview</td>
<td>125</td>
</tr>
<tr>
<td>2.6.4.2 RAPID components</td>
<td>126</td>
</tr>
<tr>
<td>2.6.4.3 Code examples</td>
<td>127</td>
</tr>
<tr>
<td>2.7 Fixed Position Events</td>
<td>129</td>
</tr>
<tr>
<td>2.7.1 Overview</td>
<td>129</td>
</tr>
<tr>
<td>2.7.2 RAPID components and system parameters</td>
<td>130</td>
</tr>
<tr>
<td>2.7.3 Code examples</td>
<td>133</td>
</tr>
<tr>
<td>2.8 Logical Cross Connections</td>
<td>135</td>
</tr>
<tr>
<td>2.8.1 Introduction to Logical Cross Connections</td>
<td>135</td>
</tr>
<tr>
<td>2.8.2 Configuring Logical Cross Connections</td>
<td>136</td>
</tr>
<tr>
<td>2.8.3 Examples</td>
<td>137</td>
</tr>
<tr>
<td>2.8.4 Limitations</td>
<td>139</td>
</tr>
<tr>
<td>2.9 RAPID Message Queue</td>
<td>140</td>
</tr>
<tr>
<td>2.9.1 Introduction to RAPID Message Queue</td>
<td>140</td>
</tr>
<tr>
<td>2.9.2 RAPID Message Queue behavior</td>
<td>141</td>
</tr>
<tr>
<td>2.9.3 System parameters</td>
<td>145</td>
</tr>
<tr>
<td>2.9.4 RAPID components</td>
<td>146</td>
</tr>
<tr>
<td>2.9.5 Code examples</td>
<td>147</td>
</tr>
</tbody>
</table>
# Table of contents

4.2.3 What happens at a collision ................................................................. 214
4.2.4 Additional information ....................................................................... 216
4.2.5 Configuration and programming facilities ..................................... 217
4.2.5.1 System parameters ........................................................................ 217
4.2.5.2 RAPID components ....................................................................... 219
4.2.5.3 Signals .......................................................................................... 220
4.2.6 How to use Collision Detection ....................................................... 221
4.2.6.1 Set up system parameters ............................................................ 221
4.2.6.2 Adjust supervision from FlexPendant .......................................... 222
4.2.6.3 Adjust supervision from RAPID program ....................................... 223
4.2.6.4 How to avoid false triggering ...................................................... 224
4.3 Collision Avoidance 3150-1 ................................................................. 225
4.4 SafeMove Assistant ............................................................................... 227
5 Motor Control ......................................................................................... 229
5.1 Independent Axis [3111-1] ..................................................................... 229
5.1.1 Overview ........................................................................................ 229
5.1.2 System parameters .......................................................................... 231
5.1.3 RAPID components ......................................................................... 232
5.1.4 Code examples ................................................................................. 233
6 RAPID Program Features ....................................................................... 235
6.1 Path Recovery [3113-1] ......................................................................... 235
6.1.1 Overview ........................................................................................ 235
6.1.2 RAPID components ......................................................................... 236
6.1.3 Store current path .............................................................................. 237
6.1.4 Path recorder .................................................................................... 239
6.2 Multitasking [3114-1] ........................................................................ 243
6.2.1 Introduction to Multitasking ............................................................ 243
6.2.2 System parameters .......................................................................... 245
6.2.3 RAPID components ......................................................................... 247
6.2.4 Communication between tasks ....................................................... 248
6.2.4.1 Persistent variables ....................................................................... 248
6.2.4.2 Waiting for other tasks ................................................................. 250
6.2.4.3 Synchronizing between tasks ....................................................... 252
6.2.4.4 Using a dispatcher ....................................................................... 254
6.2.5 Other programming issues ............................................................... 256
6.2.5.1 Share resource between tasks ....................................................... 256
6.2.5.2 Test if task controls mechanical unit .......................................... 257
6.2.5.3 taskid ......................................................................................... 258
6.2.5.4 Avoid heavy loops ....................................................................... 259
7 Communication ....................................................................................... 261
7.1 FTP&SFTP client [3116-1] ..................................................................... 261
7.1.1 Introduction to FTP&SFTP client ..................................................... 261
7.2 NFS Client [3117-1] ........................................................................... 264
7.2.1 Introduction to NFS Client ............................................................... 264
8 User Interaction Application ................................................................... 267
8.1 RobotStudio Connect [3119-1] .............................................................. 267
8.2 FlexPendant Base Apps ....................................................................... 268
8.3 FlexPendant Independent Apps ............................................................ 269
9 Engineering tools ..................................................................................... 271
9.1 RobotWare Add-In ............................................................................... 271
9.2 Path Corrections [3123-1] .................................................................... 272
9.2.1 Overview ........................................................................................ 272
9.2.2 RAPID components ....................................................................... 274
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2.3</td>
<td>Related RAPID functionality</td>
<td>275</td>
</tr>
<tr>
<td>9.2.4</td>
<td>Code example</td>
<td>276</td>
</tr>
<tr>
<td>9.3</td>
<td>Auto Acknowledge Input</td>
<td>277</td>
</tr>
</tbody>
</table>

Index 279
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

<table>
<thead>
<tr>
<th>Reference</th>
<th>Document ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product specification - OmniCore C line</td>
<td>3HAC065034-001</td>
</tr>
<tr>
<td>Product specification - OmniCore E line</td>
<td>3HAC079823-001</td>
</tr>
<tr>
<td>Operating manual - RobotStudio</td>
<td>3HAC032104-001</td>
</tr>
<tr>
<td>Operating manual - OmniCore</td>
<td>3HAC065036-001</td>
</tr>
<tr>
<td>Operating manual - Integrator's guide OmniCore</td>
<td>3HAC065037-001</td>
</tr>
<tr>
<td>Technical reference manual - RAPID Instructions, Functions and Data types</td>
<td>3HAC065038-001</td>
</tr>
<tr>
<td>Technical reference manual - RAPID Overview</td>
<td>3HAC065040-001</td>
</tr>
<tr>
<td>Technical reference manual - System parameters</td>
<td>3HAC065041-001</td>
</tr>
</tbody>
</table>

Revisions

<table>
<thead>
<tr>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Released with RobotWare 7.0.</td>
</tr>
</tbody>
</table>
| B        | Released with RobotWare 7.01. The following updates are made in this revision:  
- "Cyber security" replaced by "Cybersecurity" in entire manual.  
- Updated the section Connected Services on page 51. |

Continues on next page
<table>
<thead>
<tr>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
</table>
| C        | Released with RobotWare 7.0.2.  
The following updates are made in this revision:  
- FlexPendant terminology updated in entire manual.  
- Updated the section *Summary of Connected Services paths in Flex-Pendant* on page 56. |
| D        | Released with RobotWare 7.1.  
The following updates are made in this revision:  
- Updated the section *Connected Services* on page 51.  
- Added information about YuMi robots and Collision Detection, see *Collision detection for YuMi robots* on page 212.  
- Updated limitations for SFTP client, see *Limitations* on page 262. |
| E        | Released with RobotWare 7.2.  
The following updates are made in this revision:  
- Updated the section *Connected Services* on page 51.  
- Information about the digital output `MotSupOn` updated in section *Signals* on page 220.  
- Section *System parameters* on page 145 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 262 (FTP&SFTP client) and *Limitations* on page 264 (NFS Client). |
| F        | Released with RobotWare 7.3.  
The following updates are made in this revision:  
- Updated the section *Connected Services* on page 51. |
| G        | Released with RobotWare 7.4.  
The following updates are made in this revision:  
- Added the section *Connected Services Embedded troubleshooting logs* on page 92.  
- Updated the section *Connected Services* on page 51.  
- Updated information regarding UTF-8, in *Raw data communication* on page 121. |
| H        | Released with RobotWare 7.5.  
- Updated limitation for *Collision Avoidance 3150-1* on page 225. |
| J        | Released with RobotWare 7.6.  
- Added the section *Auto Acknowledge Input* on page 277.  
- Updated limitation regarding lead-through, see *Overview of World Zones* on page 205.  
- Added the section *SafeMove Assistant* on page 227. |
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.

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

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

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:

<table>
<thead>
<tr>
<th>Product classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RobotWare-OS</td>
<td>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. For a description of RobotWare-OS, see the product specification for the robot controller.</td>
</tr>
<tr>
<td>RobotWare options</td>
<td>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. Note Not all RobotWare options are described in this manual. Some options are more comprehensive and are therefore described in separate manuals.</td>
</tr>
<tr>
<td>Process application options</td>
<td>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. The process application options are all described in separate manuals.</td>
</tr>
<tr>
<td>RobotWare Add-ins</td>
<td>A RobotWare Add-in is a self-contained package that extends the functionality of the robot system. 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. 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 <a href="http://www.abb.com/contacts">www.abb.com/contacts</a>.</td>
</tr>
</tbody>
</table>
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:

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

Note

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

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

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>The data type byte represent a decimal value between 0 and 255.</td>
</tr>
</tbody>
</table>

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*.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitSet</td>
<td>BitSet is used to set a specified bit to 1 in a defined byte data.</td>
</tr>
<tr>
<td>BitClear</td>
<td>BitClear is used to clear (set to 0) a specified bit in a defined byte data.</td>
</tr>
</tbody>
</table>

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*.

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

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.
2.1.2.3 Bit functionality example

Program code

```plaintext
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`. 
2.1.3.2 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*.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datapos</td>
<td><em>datapos</em> is the enclosing block to a data object (internal system data) retrieved with the function <em>GetNextSym</em>.</td>
</tr>
</tbody>
</table>

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*.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetDataSearch</td>
<td><em>SetDataSearch</em> is used together with <em>GetNextSym</em> to retrieve data objects from the system.</td>
</tr>
<tr>
<td>GetDataVal</td>
<td><em>GetDataVal</em> 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 <em>GetNextSym</em>.</td>
</tr>
<tr>
<td>SetDataVal</td>
<td><em>SetDataVal</em> 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 <em>GetNextSym</em>.</td>
</tr>
<tr>
<td>SetAllDataVal</td>
<td><em>SetAllDataVal</em> make it possible to set a new value to all data objects of a certain type that match the given grammar.</td>
</tr>
</tbody>
</table>

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*.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getNextSym</td>
<td><em>getNextSym</em> (<em>Get Next Symbol</em>) is used together with <em>SetDataSearch</em> to retrieve data objects from the system.</td>
</tr>
</tbody>
</table>
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
2.1.4 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*.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AliasIO</td>
<td><em>AliasIO</em> 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 <em>AliasIO</em> must be run before any use of the actual signal.</td>
</tr>
</tbody>
</table>

Functions

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

<table>
<thead>
<tr>
<th>Purpose</th>
<th>What is included</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>Configuration functionality includes the instructions: <code>ReadCfgData</code>, <code>WriteCfgData</code>, and <code>WarmStart</code>.</td>
</tr>
</tbody>
</table>
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*.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadCfgData</td>
<td>ReadCfgData is used to read one attribute of a named system parameter</td>
</tr>
<tr>
<td></td>
<td>(configuration data).</td>
</tr>
<tr>
<td>WriteCfgData</td>
<td>WriteCfgData is used to write one attribute of a named system parameter</td>
</tr>
<tr>
<td></td>
<td>(configuration data).</td>
</tr>
<tr>
<td>WarmStart</td>
<td>WarmStart is used to restart the controller at run time.</td>
</tr>
<tr>
<td></td>
<td>This is useful after changing system parameters with the instruction</td>
</tr>
<tr>
<td></td>
<td>WriteCfgData.</td>
</tr>
</tbody>
</table>

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 \textit{cal\_offset} for \texttt{rob1\_1} is read, increased by 0.2 mm and then written back. To make this change take effect, the controller is restarted.

\begin{verbatim}
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;
\end{verbatim}
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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFRestart</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
2.1.6.3 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`. 
2.1.7.2 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*.

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

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*.

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

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.

```plaintext
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.

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

Continues on next page
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.
2.1.8.2 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.

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

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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPers</td>
<td>IPers (Interrupt Persistent) is used to order an interrupt to be generated each time the value of a persistent variable is changed.</td>
</tr>
<tr>
<td>IError</td>
<td>IError (Interrupt Errors) is used to order an interrupt to be generated each time an error occurs.</td>
</tr>
<tr>
<td>GetTrapData</td>
<td>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.</td>
</tr>
<tr>
<td>ReadErrData</td>
<td>ReadErrData is used in trap routines generated by the instruction IError. ReadErrData read the information obtained by GetTrapData.</td>
</tr>
<tr>
<td>ErrRaise</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

```plaintext
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.

```plaintext
VAR intnum err_interrupt;
VAR trapdata err_data;
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;

PROC main()
    CONNECT err_interrupt WITH trap_err;
    IError COMMON_ERR, TYPE_ERR, err_interrupt;
    ...
    a:=3;
    b:=0;
    c:=a/b;
    ...
    IDelete err_interrupt;
ENDPROC

TRAP trap_err
    GetTrapData err_data;
    ReadErrData err_data, err_domain, err_number, err_type;
    SetGO go_err1, err_domain;
    SetGO go_err2, err_number;
ENDTRAP
```
2.1.9 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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BookErrNo</td>
<td>BookErrNo is used to define a new RAPID system error number.</td>
</tr>
<tr>
<td>TextTabInstall</td>
<td>TextTabInstall is used to install a text table in the system.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextTabFreeToUse</td>
<td>TextTabFreeToUse is used to test whether the text table name is free to use (not already installed in the system).</td>
</tr>
<tr>
<td>TextTabGet</td>
<td>TextTabGet is used to get the text table number of a user defined text table.</td>
</tr>
<tr>
<td>TextGet</td>
<td>TextGet is used to get a text string from the system text tables.</td>
</tr>
</tbody>
</table>
2 RobotWare-OS

2.1.9.3 User message functionality examples

---

### Book error number

This example shows how to add a new error number.

```plaintext
VAR intnum sig1int;

!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 di1=1
IF di1=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.

```plaintext
VAR num text_res_no;

PROC install_text()
!Test if text_table_name is already installed
IF TextTabFreeToUse("text_table_name") THEN
    !Install the table from the file HOME:/language/en/text_file.xml
    TextTabInstall "HOME:/language/en/text_file.xml";
ENDIF
!Assign the text table number for text_table_name to text_res_no
text_res_no := TextTabGet("text_table_name");
ENDPROC
...
!Write error message with two strings from the table text_res_no
```
ErrWrite TextGet(text_res_no, 1), TextGet(text_res_no, 2);
2.1.9.4 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.

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

Example of text table file

```xml
<?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`.
2.1.10.2 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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetSysData</td>
<td>SetSysData activates (or changes the current active) tool, work object, or payload for the robot.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgName</td>
<td>ArgName is used to get the name of the original data object for the current argument or the current data.</td>
</tr>
<tr>
<td>IsStopStateEvent</td>
<td>IsStopStateEvent returns information about the movement of the program pointer.</td>
</tr>
</tbody>
</table>
2.1.10.3 RAPID support functionality examples

Activate tool

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

```rapid
!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:

```rapid
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".

```rapid
VAR num my_nbr := 5;
proc1 my_nbr;

PROC proc1 (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.

```rapid
IF IsStopStateEvent (\PPMoved) = TRUE THEN
TPWrite "The program pointer has been moved.\n"
ENDIF
```
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 50.

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*.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISignalAI</td>
<td>Defines the values of an analog input signal, for which an interrupt routine shall be called. 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.</td>
</tr>
<tr>
<td>ISignalAO</td>
<td>Defines the values of an analog output signal, for which an interrupt routine shall be called. 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.</td>
</tr>
</tbody>
</table>

#### 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 ai1.

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 ai1_warning;
VAR intnum ai1_exeeded;

PROC main()
    CONNECT ai1_warning WITH temp_warning;
    CONNECT ai1_exeeded WITH temp_exeeded;
    ISignalAI ai1, AIO_BETWEEN, 130, 120, 0.5, \DPos, ai1_warning;
    ISignalAI \Single, ai1, AIO_ABOVE_HIGH, 130, 120, 0, ai1_exeeded;
...
    IDelete ai1_warning;
    IDelete ai1_exeeded;
ENDPROC

TRAP temp_warning
    TPWrite "Warning: Temperature is \Num:=ai1;"
ENDTRAP

TRAP temp_exceeded
    TPWrite "Temperature is too high";
    Stop;
ENDTRAP
2.3 Connected Services

2.3.1 Overview

Description

Connected Services is a functionality for ABB robot controllers to connect to ABB Ability™ Connected Services Cloud by using 3G, WiFi, 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 for Connected Services 1.0, Connected Services portal for Connected Services 2.0, 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.

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.

Continues on next page
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 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 Ability™ 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 Ability™ Cloud to keep the customer network isolated from any external Internet access. The following figures describe these concepts.

For Connected Services 1.0:

For Connected Services 2.0:
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:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Check controller S/N and internet connectivity</td>
</tr>
<tr>
<td>1a</td>
<td>Enable CSE and set up connectivity configuration</td>
</tr>
<tr>
<td>2a</td>
<td>CS connectivity in place</td>
</tr>
<tr>
<td>2b</td>
<td>Low poll for registration</td>
</tr>
<tr>
<td>2c</td>
<td>Registration not trusted (get reg code)</td>
</tr>
<tr>
<td>2d</td>
<td>Display registration code</td>
</tr>
<tr>
<td>3a</td>
<td>Get registration code</td>
</tr>
<tr>
<td>3b</td>
<td>Give controller S/N and registration code</td>
</tr>
<tr>
<td>3c</td>
<td>Select controller S/N in SA and register with registration code</td>
</tr>
<tr>
<td>3d</td>
<td>Registration trusted (client certificate)</td>
</tr>
<tr>
<td>4</td>
<td>Connected and registered secure CS session</td>
</tr>
</tbody>
</table>
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 62.

Connected Services connectivity
1. The software agent connects to ABB Ability™ 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 Connected Services 1.0 and in MyABB/Register my robot controller for Connected Services 2.0 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.

Note
With Connected Services 2.0 a second certificate is also installed to trust ABB Ability Cloud.

3. Connected Services is now actively running on the robot controller.

Note
In case of Power On Connect with 3G Connected Services Gateway and production registration, all these processes are 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 Ability™ > 3G Connection
- CS Gateway WiFi: Settings > ABB Ability™ > WiFi Connection
- CS Gateway Wired: Settings > ABB Ability™ > Wired Connection

### Status

In FlexPendant the Connected Services status are available in:

- Ability Network: Settings > ABB Ability™ > Network Status
- CS Gateway: Settings > ABB Ability™ > Connectivity Status
- Connected Services summary: QuickSet > ABB Ability™
- Connected Services details: Settings > ABB Ability™ > 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
2.3.5 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

### 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/WiFi): Not implemented
- Details Logs: Controller > Properties > Save Diagnostic
### 2.3.6 Configuration - 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*.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Enables or disables the Connected Services connection between the controller and the server.</td>
</tr>
<tr>
<td>Connection Type</td>
<td>Indicates if the communication is done on Ability™ (ABB Connected Services Gateway solution), Public, or Custom network.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>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 <em>Operating manual - Integrator’s guide OmniCore</em>.</td>
</tr>
<tr>
<td>Internet Gateway IP</td>
<td>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.</td>
</tr>
<tr>
<td>Internet DNS IP</td>
<td>Defines the internet DNS IP of the connected network. This is valid for the Connection Type Custom. Forces the DNS to use for connectivity.</td>
</tr>
<tr>
<td>Proxy Used</td>
<td>Defines if a proxy is used to access the internet and its name/address, port, and authentication.</td>
</tr>
<tr>
<td>Proxy Auth</td>
<td>Defines the proxy authentication type. Basic will use HTTP basic authentication including user and password. None will not use any.</td>
</tr>
<tr>
<td>Server Polling</td>
<td>Defines the frequency of polling for specific synchronization with the server. The available values are slow and fast. For details about the behavior of events for server polling, see <em>Description of behavior of events for server polling on page 86</em>.</td>
</tr>
<tr>
<td>Debug Mode</td>
<td>Enables extensive logging for debugging the issues.</td>
</tr>
<tr>
<td>Trace Level</td>
<td>Defines the level of logging if Debug Mode is enabled.</td>
</tr>
</tbody>
</table>
**Parameter** | **Description**
--- | ---
Connected Services Mode | Defines the compatibility for different robot controller’s data format, cloud solution, and specific features. Following are the available modes:
- 1.0 IRC5 Compatibility
- 2.0 Omnicore [Preview]

Note
By default, the 1.0 IRC5 Compatibility mode is enabled.

Note
2.0 Omnicore [Preview] can be configured only from RobotStudio. As Connected Services 2.0 is still in preview mode, please synchronize, before using, with your local ABB Customer Service for usage, status, and capability.

Customer Storage | Defines whether the data needs to be pushed to external network disk or not (None, Disk) and MQTT. The disk path can be defined separately. This parameter is valid only for CSE 2.0.

Disk Path | Defines the path to the external disk if Disk is selected. This parameter is valid only for CSE 2.0.

Scheme | Defines the type of scheme or protocol used for MQTT communication. This parameter is valid only for CSE 2.0.

Web Socket | Defines whether the scheme uses web socket connection or not. This parameter is valid only for CSE 2.0.

Host | Defines the MQTT host IP. This parameter is valid only for CSE 2.0.

Port | Defines the MQTT port number. This parameter is valid only for CSE 2.0.

URL | Defines the MQTT topic to publish data. This parameter is valid only for CSE 2.0.

Auth Type | Defines the authentication used for the MQTT connection. The authentication type is basic authentication with user/password or Client Certificate. This parameter is valid only for CSE 2.0.

Certificate Path | Defines the location to client certificate path. This parameter is valid only for CSE 2.0.

Key Path | Defines the location to client private key file path. This parameter is valid only for CSE 2.0.

Server Check | Defines whether the server certificate needs to be validated against CA. This parameter is valid only for CSE 2.0.

CA Path | Defines the path to certificate authority or root certificate. This parameter is valid only for CSE 2.0.

Force Zip | Defines whether the data need to be compressed for publishing. This parameter is valid only for CSE 2.0. (Not yet implemented.)

Continued
### 2 RobotWare-OS

#### 2.3.6 Configuration - system parameters

Continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time Out</strong></td>
<td>Defines the MQTT connection timeout. This parameter is valid only for CSE 2.0. (Not yet implemented.)</td>
</tr>
<tr>
<td><strong>Keep Alive</strong></td>
<td>Defines the MQTT keep alive time. This parameter is valid only for CSE 2.0. (Not yet implemented.)</td>
</tr>
<tr>
<td><strong>Reconnect Period</strong></td>
<td>Defines the MQTT reconnect interval. This parameter is valid only for CSE 2.0. (Not yet implemented.)</td>
</tr>
<tr>
<td><strong>Retry</strong></td>
<td>Defines the retry count for MQTT connection. This parameter is valid only for CSE 2.0. (Not yet implemented.)</td>
</tr>
</tbody>
</table>

**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](image)

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](image)

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

These parameters belong to the topic Communication and the type IP Routing. In some cases it is necessary to define some routing parameters to indicate which specific external device is used as a gateway to access the Internet on customer network. By default, an IP route is created based on the gateway defined on the WAN Port. But it is possible to add a specific route if the default gateway should not be used. For more details, see Type IP Route in Technical reference manual - System parameters.

Note

If the Internet Gateway is not the main Gateway, the traffic to rseprod.abb.com and the DNS must be defined as additional routes. For example, if Internet Gateway has IP address 100.100.100.22, rseprod.abb.com has IP address 138.227.175.43 (verify by nslookup) and DNS has IP address 8.8.8.8, then you must define the following two routes:

• Route 138.227.175.43/31 to 100.100.100.22
• Route 8.8.8.8/31 to 100.100.100.22
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, 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 Ability 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:

- Ability
- Public
- Custom

Configuring the connection type Ability

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

Note

The connection type Ability is enabled by default.

Use the following procedure to configure the connection type Ability using FlexPendant:

1. On the start screen, tap Settings, and then select ABB Ability 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 Ability.

Note

The Ability network can be configured based on the available CS Gateway (3G, 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 76.

Continues on next page
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 67.

**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 Ability** 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 76*.

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 Ability** 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 **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 76.*
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 Ability 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.

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

- **Ability**
- **Public**
- **Custom**

### Configuring the connection type Ability

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

**Note**

The connection type **Ability** is enabled by default.

Use the following procedure to configure the connection type **Ability** 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 **Ability Network**.

   **Note**

   Ability Network can be configured based on the available CS Gateway (3G, 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 76*.

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

Continues on next page
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 74.*

**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 76.* 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
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 76.*

   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.
   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.

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 Ability™ > 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.

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Possible values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Agreement</td>
<td>To verify that the controller is associated to the expected service agreement.</td>
<td>&quot;Name of the service agreement&quot; &quot;.&quot;</td>
<td>SA_FR12_16</td>
</tr>
<tr>
<td>Customer name</td>
<td>To verify that the controller is associated to the expected service agreement.</td>
<td>&quot;Customer Name of the service agreement&quot; &quot;.&quot;</td>
<td>ABB Robotics</td>
</tr>
<tr>
<td>Country</td>
<td>To verify that the controller is associated to the expected service agreement.</td>
<td>&quot;Country of the service agreement&quot; &quot;.&quot;</td>
<td>France</td>
</tr>
<tr>
<td>ABB server</td>
<td>Displays the type of the server connected services is connected.</td>
<td>Robotics Cloud</td>
<td>Robotics Cloud</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Possible values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Displays the current status to see whether there is a need to navigate to the Connectivity page or Registration page.</td>
<td>For a description of values, see CSE Status on page 83.</td>
<td>Active</td>
</tr>
<tr>
<td>Connection Status</td>
<td>Displays the status of communication with the server and the type of error.</td>
<td>For a description of values, see CSE Connection Status on page 84.</td>
<td>Connected</td>
</tr>
<tr>
<td>Last updated</td>
<td>Displays the relative time since the information on the Connectivity page has been generated.</td>
<td>&quot;HH:MM:SS ago&quot;</td>
<td></td>
</tr>
<tr>
<td>Hardware gateway</td>
<td>Displays the type of hardware gateway and connection IP.</td>
<td>DSQC 1039 3G Connected on IP: 192.168.126.1</td>
<td></td>
</tr>
<tr>
<td>Server name</td>
<td>Displays the name of the server that software agent is configured with.</td>
<td>&quot;&quot; Server name</td>
<td></td>
</tr>
<tr>
<td>Server IP</td>
<td>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.</td>
<td>&quot;&quot; Server IP</td>
<td></td>
</tr>
<tr>
<td>Server certificate name</td>
<td>Displays the server certificate name information.</td>
<td>&quot;&quot; Server name Untrusted (Server)</td>
<td></td>
</tr>
</tbody>
</table>

Continues on next page
### Field Description Possible values Example

**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

**Controller time** Displays the controller date and time details. 16-01-08 13:52:33

| Note | It is important to set the correct time in the controller as this is needed for the certificate process. |

**Connection type** Displays the type of network connection used. Ability

**Public network information** Displays the network information for the Public port. NoIP Static DHCP NoIP Static DHCP

| DHCP: 10.140.198.55/255.255.255.0/10.140.198.1/ plugged |

**Ability network information** Displays the network information for the Ability 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

---

**Note**

- It is important to set the correct time in the controller as this is needed for the certificate process.
### Registration page

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Possible values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status</strong></td>
<td>Displays the current status to see whether there is a need to navigate to the Server connection page or Registration page.</td>
<td>For a description of values, see <a href="#">CSE Status on page 83.</a></td>
<td>Active</td>
</tr>
<tr>
<td><strong>Registration Status</strong></td>
<td>Displays the registration status.</td>
<td>For a description of values, see <a href="#">CSE Registration Status on page 85.</a></td>
<td>Register with code in MyRobot</td>
</tr>
<tr>
<td><strong>Registration code</strong></td>
<td>Displays the registration code. This code can be used to register using MyRobot.</td>
<td></td>
<td>456735</td>
</tr>
</tbody>
</table>

### Advanced page

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Possible values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Last HTTP message</strong></td>
<td>Displays the last message sent.</td>
<td>Register CheckRegistered LogMessage GetMessage GetConfig SendGetSpecificCode DownLoadFile AcknowledgeMessage BoxUpload GetRSEAgreementInformation SendDeviceInformation RequestClientCertificate RenewClientCertificate periodicDeviceUpdateInformation</td>
<td>GetMessage</td>
</tr>
<tr>
<td><strong>Last HTTP message time</strong></td>
<td>Displays the date and time when the last message was sent.</td>
<td></td>
<td>Sent 00:01:28 ago</td>
</tr>
<tr>
<td><strong>Last HTTP error</strong></td>
<td>Displays the HTTP error when the last message was sent and the message ID if 4XX.</td>
<td>Not Available - if no error Error HTTP XXX + Message</td>
<td>Not Available</td>
</tr>
</tbody>
</table>
### 2.3.9 Connected Services information

**Continued**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Possible values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next message</td>
<td>Displays the next message to send and the date to send the message.</td>
<td></td>
<td>GetMessage in 70 seconds</td>
</tr>
<tr>
<td>Last command</td>
<td>Displays the last command received from server.</td>
<td>Not Available</td>
<td>Not Available</td>
</tr>
<tr>
<td>Restart counter</td>
<td>Displays the number of times the software Agent been auto-restarted. This is used to see if watchdog has restarted.</td>
<td>0-N If not Enabled, then display 0</td>
<td>2</td>
</tr>
<tr>
<td>Connector version</td>
<td>Displays the currently running connector version information.</td>
<td></td>
<td>1.0.0</td>
</tr>
<tr>
<td>Data Collector status</td>
<td>Displays the status of the data collector.</td>
<td>Started</td>
<td>Started</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not Started</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Downloaded</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Download Failed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Failed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stopped</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disabled</td>
<td></td>
</tr>
<tr>
<td>Last registered</td>
<td>Displays the last registration date and time to server of controller.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Last connected</td>
<td>Displays the last date and time on which the controller detects a successful connection with the server.</td>
<td>11-07-2020 08:33:35</td>
<td></td>
</tr>
<tr>
<td>Server polling</td>
<td>Displays the Server polling configuration. Server polling is related to the calculation of connection cost.</td>
<td>Fast Slow</td>
<td>Fast</td>
</tr>
<tr>
<td>Bytes sent/received</td>
<td>Displays the number of bytes sent/received.</td>
<td></td>
<td>17.0KB/24.17KB</td>
</tr>
<tr>
<td>Connected Services mode</td>
<td>Displays the status of the connected services mode.</td>
<td>For a description of values, see <a href="#">CSE Mode on page 85</a>.</td>
<td>Active Mode</td>
</tr>
<tr>
<td>Server errors</td>
<td>Displays a count of the following servers errors:</td>
<td>0-N for each server error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Connection errors</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 Connection not available errors</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 Authentication errors</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 Request errors</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 Timeout errors</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 Proxy errors</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 Unknown errors</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Root certificate issuer</td>
<td>Displays the name of the root certificate issuer.</td>
<td></td>
<td>Baltimore Cyber-Trust Root DigiCert Global Root CA</td>
</tr>
</tbody>
</table>
### Ability page

The Ability page provides advanced information about Ability connection, certificate information, and also the data transaction details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Possible values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ability connection state</td>
<td>Displays the status of Ability cloud connection.</td>
<td>For a list of possible values and description, see <a href="#">Ability status on page 85</a>.</td>
<td>Connected to IoT Hub</td>
</tr>
<tr>
<td>Ability connection error count</td>
<td>Displays the number of connection error with Ability cloud connection.</td>
<td>0-N</td>
<td>2</td>
</tr>
<tr>
<td>Ability Device ID</td>
<td>Displays the device id for the Ability cloud connection.</td>
<td></td>
<td>1ca930c7-e77f-4265-8005-0ee493b84eeb</td>
</tr>
<tr>
<td>Ability Client certificate device</td>
<td>Displays the common name of client certificate used for Ability cloud connection.</td>
<td></td>
<td>1ca930c7-e77f-4265-8005-0ee493b84eeb</td>
</tr>
<tr>
<td>Ability Client certificate issuer</td>
<td>Displays the name of the Ability client certificate issuer.</td>
<td>ABB Ability(tm) Issuing CA</td>
<td></td>
</tr>
<tr>
<td>Ability Client certificate valid from</td>
<td>Displays the date and time from which the Ability client certificate is valid.</td>
<td>-1 - if there is no data Date and time (if data is updated)</td>
<td>Jul 2 00:00:00 2020 GMT</td>
</tr>
<tr>
<td>Ability Client certificate valid until</td>
<td>Displays the date and time until which the Ability client certificate is valid.</td>
<td>-1 - if there is no data Date and time - if data is updated</td>
<td>Jul 3 23:59:59 2020 GMT</td>
</tr>
<tr>
<td>Ability Client certificate serial number</td>
<td>Displays the Ability client certificate serial number.</td>
<td></td>
<td>063433724FA28636BCB72916FF472CF6</td>
</tr>
<tr>
<td>DPS Server Name</td>
<td>Displays the DPS server name.</td>
<td>global azure-devices-provisioning.net</td>
<td></td>
</tr>
<tr>
<td>DPS Server IP</td>
<td>Displays the resolved IP address of the DPS server.</td>
<td></td>
<td>52.163212.39</td>
</tr>
<tr>
<td>DPS Scope ID</td>
<td>Displays the DPS Scope Id of the cloud configuration.</td>
<td></td>
<td>0neOOA3934</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Possible values</td>
<td>Example</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>DPS Root certificate issuer</td>
<td>Displays the name of the DPS root certificate issuer.</td>
<td></td>
<td>Microsoft IT TLS CA 4</td>
</tr>
<tr>
<td>DPS Root certificate subject</td>
<td>Displays the subject of the DPS root certificate.</td>
<td></td>
<td>.azure-devices-provisioning.net</td>
</tr>
<tr>
<td>DPS enrolment group ID</td>
<td>Displays the group ID of the DPS enrollment.</td>
<td></td>
<td>ABBAbilitytmIssuingCA</td>
</tr>
<tr>
<td>IoT Hub Server Name</td>
<td>Displays the name of the Ability IoT hub Server.</td>
<td></td>
<td>AbiGtylthiro2Eun-Bdv.azure-devices.net</td>
</tr>
<tr>
<td>IoT Hub Server IP</td>
<td>Displays the resolved IP address of Ability IoT hub server.</td>
<td>Resolved IP address IP not resolved</td>
<td>13.79.172.43</td>
</tr>
<tr>
<td>IoT Hub blob storage name</td>
<td>Displays the name of the Ability IoT hub blob storage.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IoT Hub blob storage IP</td>
<td>Displays the resolved IP address of Ability IoT hub blob storage server.</td>
<td>Resolved IP address IP not resolved</td>
<td>52.239.137.68</td>
</tr>
<tr>
<td>Last sent data type</td>
<td>Displays the last sent information model data type.</td>
<td></td>
<td>abb.robotics.motionDeviceSystem@l</td>
</tr>
<tr>
<td>Last message sent</td>
<td>Displays the last successfully sent telemetry data.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Last message sent time</td>
<td>Displays the date and time of last successful sent telemetry message.</td>
<td>-1 -if there is no data Date and time (if data is updated)</td>
<td>Thu Jul 2 17:45:53 2020</td>
</tr>
<tr>
<td>Last sent Owner ID</td>
<td>Displays the owner id of the last sent information model.</td>
<td></td>
<td>34554e14-71 c6-49c0-a2a8-819e 1a6c 17cd</td>
</tr>
<tr>
<td>Last sent Object ID</td>
<td>Displays the object id of the last sent information model.</td>
<td></td>
<td>f827f6f-68b3-4785-adfd-f1e87f06bd</td>
</tr>
<tr>
<td>Last sent Tenant ID</td>
<td>Displays the tenant id of the last sent information model.</td>
<td></td>
<td>8b5a949f69b-41a5b198-1cbd5caad30c</td>
</tr>
<tr>
<td>Last message attempted</td>
<td>Displays the last telemetry message that is attempted to send.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Last message failed time</td>
<td>Displays the date and time of last failed to send telemetry message.</td>
<td>-1 -if there is no data Date and time (if data is updated)</td>
<td></td>
</tr>
<tr>
<td>Telemetry message count</td>
<td>Displays the count of successfully sent telemetry messages.</td>
<td></td>
<td>0-N</td>
</tr>
<tr>
<td>Telemetry bytes sent/received</td>
<td>Displays the amount of telemetry data uploaded and downloaded.</td>
<td></td>
<td>4146/1086 B</td>
</tr>
<tr>
<td>File upload count</td>
<td>Displays the number of files successfully uploaded to IoT blob storage.</td>
<td></td>
<td>0-N</td>
</tr>
</tbody>
</table>
### Data collectors page

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Possible values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files bytes sent/received</td>
<td>Displays the amount of data used for uploading and downloading of files.</td>
<td>0/0 B</td>
<td></td>
</tr>
<tr>
<td>Telemetry failed count</td>
<td>Displays the count of failed to send telemetry messages.</td>
<td>0-N</td>
<td>1</td>
</tr>
<tr>
<td>File upload failed count</td>
<td>Displays the number of files failed to upload to IoT blob storage.</td>
<td>0-N</td>
<td>0</td>
</tr>
</tbody>
</table>

**Note**

IRC5 Compatibility is for CSE 1.0 while Motion Data Collector and Robot System Data Collector is for CSE 2.0.

---

### Description of values in Connected Services information

**CSE Status**

The following table gives the information of CSE status:

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE_FAILED</td>
<td>Failed</td>
<td>Connected Services status is failed.</td>
</tr>
<tr>
<td>BASE_INITIALIZING</td>
<td>Initializing</td>
<td>Connected Services status is initializing.</td>
</tr>
<tr>
<td>BASE_ACTIVE</td>
<td>Active</td>
<td>Connected Services status is active.</td>
</tr>
<tr>
<td>BASE_CONNECT</td>
<td>Trying to connect</td>
<td>Connected Services status is trying to connect.</td>
</tr>
<tr>
<td>BASE_SHUTDOWN</td>
<td>Shutdown mode</td>
<td>Connected Services status is shutdown mode.</td>
</tr>
<tr>
<td>UNKNOWN_STATUS</td>
<td>Unknown</td>
<td>Connected Services status is unknown.</td>
</tr>
<tr>
<td>BASE_SLEEP</td>
<td>Sleep</td>
<td>Connected Services status is sleep mode.</td>
</tr>
</tbody>
</table>
### CSE Connection Status

The following table gives the information of CSE connection status:

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER_REQUEST_TIMEOUT_ERROR</td>
<td>Request timed out</td>
<td>Connection status request is timed out.</td>
</tr>
<tr>
<td>SERVER_CONNECTED</td>
<td>Connected</td>
<td>Connection status in connected.</td>
</tr>
<tr>
<td>SERVER_NETWORK_ERROR</td>
<td>Server not reachable</td>
<td>Connection status server is not reachable.</td>
</tr>
<tr>
<td>SERVER_AUTH_ERROR</td>
<td>Server not authenticated</td>
<td>Connection status server is not authenticated.</td>
</tr>
<tr>
<td>SERVER_CERT_ERROR</td>
<td>Server certification verification error</td>
<td>Connection status server is certification verification error.</td>
</tr>
<tr>
<td>SERVER_HTTP_ERROR</td>
<td>Server error (HTTP)</td>
<td>Connection status server is HTTP error.</td>
</tr>
<tr>
<td>SERVER_PROXY_AUTH_ERROR</td>
<td>Proxy Authentication Required</td>
<td>Connection status proxy authentication error is required.</td>
</tr>
<tr>
<td>SERVER_REQUEST_ERROR</td>
<td>Request error</td>
<td>Connection status is request error.</td>
</tr>
<tr>
<td>SERVER_PROXY_CONN_ERROR</td>
<td>Proxy Connection Error</td>
<td>Connection status proxy connection error.</td>
</tr>
<tr>
<td>SERVER_GATEWAY_DISABLED</td>
<td>CS Gateway disabled</td>
<td>Connection status Gateway is disabled.</td>
</tr>
<tr>
<td>SERVER_GATEWAY_WAITING</td>
<td>Identifying CS Gateway</td>
<td>Connection status Gateway is identifying.</td>
</tr>
<tr>
<td>SERVER_GATEWAY_WAITING_3G</td>
<td>Waiting 3G connectivity</td>
<td>Connection status Gateway is waiting for 3G connectivity.</td>
</tr>
<tr>
<td>SERVER_GATEWAY_WAITING_WIFI</td>
<td>Waiting Wi-Fi connectivity</td>
<td>Connection status Gateway is waiting for Wi-Fi connectivity.</td>
</tr>
<tr>
<td>SERVER_GATEWAY_CONNECTED</td>
<td>CS Gateway connected</td>
<td>Connection status Gateway is connected.</td>
</tr>
<tr>
<td>SERVER_LOOKUP_IN_PROGRESS</td>
<td>Host name lookup in progress</td>
<td>Connection status Gateway host name lookup in progress.</td>
</tr>
<tr>
<td>WAITING_GATEWAY_IP</td>
<td>Waiting for Gateway IP</td>
<td>Connection status Gateway is waiting for Gateway IP.</td>
</tr>
<tr>
<td>SERVER_DNS_RESOLUTION_FAIL</td>
<td>DNS resolution failed</td>
<td>Connection status Gateway has server DNS resolution failure.</td>
</tr>
<tr>
<td>SERVER_GW_PING_TIMEOUT</td>
<td>Gateway ping timeout</td>
<td>Connection status Gateway server ping is time out.</td>
</tr>
<tr>
<td>DNS_NOT_PINGABLE</td>
<td>DNS not pingable</td>
<td>Connection status Gateway has DNS that is not pingable.</td>
</tr>
<tr>
<td>AUTHN_ERROR</td>
<td>Authentication error</td>
<td>Connection status Gateway having authentication error.</td>
</tr>
<tr>
<td>RESTART_ERROR</td>
<td>Waiting to start after restart error</td>
<td>Connection status Gateway is waiting to start after restart error.</td>
</tr>
</tbody>
</table>
## 2.3.9 Connected Services information

### CSE Registration Status

The following table gives the information of CSE registration status:

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REG_REGISTERED</td>
<td>Registered</td>
<td>Registration status is registered.</td>
</tr>
<tr>
<td>REG_IN_PROGRESS</td>
<td>Registration in progress</td>
<td>Registration status is in progress.</td>
</tr>
<tr>
<td>REG_REGISTER</td>
<td>Registration with code in</td>
<td>Registration status is registered with code in MyRobot.</td>
</tr>
<tr>
<td></td>
<td>MyRobot</td>
<td></td>
</tr>
<tr>
<td>REG_DISABLED</td>
<td>Registration disabled</td>
<td>Registration status is disabled.</td>
</tr>
<tr>
<td>REG_FAILED</td>
<td>Failed</td>
<td>Registration status is failed.</td>
</tr>
</tbody>
</table>

### CSE Mode

The following table gives the information of CSE mode:

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

### Ability status

The following table gives the information of Ability statuses:

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_CONFIG_PEND</td>
<td>Config pending</td>
<td>Ability configuration is pending.</td>
</tr>
<tr>
<td>STATUS_CONFIG_DONE</td>
<td>Config updated</td>
<td>Ability configuration is updated.</td>
</tr>
<tr>
<td>STATUS_CLIENT_CERT_PEND</td>
<td>Client certificate pending</td>
<td>Waiting to receive a new Ability certificate.</td>
</tr>
<tr>
<td>STATUS_CLIENT_CERT_UPDATED</td>
<td>Client certificate updated</td>
<td>Ability Client certificate is updated.</td>
</tr>
<tr>
<td>STATUS_DPS_INITIATED</td>
<td>DPS Initiated</td>
<td>DPS connectivity is initiated.</td>
</tr>
<tr>
<td>STATUS_IOT_HUB_ASSIGNED</td>
<td>IoT Hub assigned</td>
<td>IoT Hub is assigned.</td>
</tr>
<tr>
<td>STATUS_DPS_FAILED</td>
<td>DPS failed</td>
<td>DPS connectivity is failed.</td>
</tr>
<tr>
<td>STATUS_IOT_HUB_CONNECTED</td>
<td>Connected to IoT Hub</td>
<td>Connected to IoT Hub.</td>
</tr>
</tbody>
</table>

Continues on next page
### Data Collection Status

The following table gives the information of data collection status:

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIFIC_STATUS_NONE</td>
<td>None</td>
<td>Data collection status is none.</td>
</tr>
<tr>
<td>SPECIFIC_STATUS_DOWNLOADED</td>
<td>Download</td>
<td>Data collection status is downloaded.</td>
</tr>
<tr>
<td>SPECIFIC_STATUS_DOWNLOAD_FAILED</td>
<td>Download Failed</td>
<td>Data collection status downloaded failed.</td>
</tr>
<tr>
<td>SPECIFIC_STATUS_STARTED</td>
<td>Started</td>
<td>Data collection status is started.</td>
</tr>
<tr>
<td>SPECIFIC_STATUS_STOPPED</td>
<td>Stopped</td>
<td>Data collection status is stopped.</td>
</tr>
<tr>
<td>SPECIFIC_STATUS_FAILED</td>
<td>Failed</td>
<td>Data collection status is failed.</td>
</tr>
</tbody>
</table>

### Description of behavior of events for server polling

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

<table>
<thead>
<tr>
<th>Events</th>
<th>Slow</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check Registration</td>
<td>30 min</td>
<td>10 min</td>
</tr>
<tr>
<td>Send HardWare Information (before registra-</td>
<td>30 min</td>
<td>10 min</td>
</tr>
<tr>
<td>tion) (Periodic polling is send only if there is any change.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send HardWare Information (after registra-</td>
<td>24 h</td>
<td>24 h</td>
</tr>
<tr>
<td>tion)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send Alive message</td>
<td>50 min</td>
<td>50 min</td>
</tr>
<tr>
<td>Check for Module Update</td>
<td>8 h</td>
<td>4 h</td>
</tr>
<tr>
<td>Send Dynamic Information</td>
<td>4 h</td>
<td>4 h</td>
</tr>
<tr>
<td>Internal Pending Command</td>
<td>10 min</td>
<td>1 min</td>
</tr>
<tr>
<td>Restart after Unregistration</td>
<td>Restart watchdog (4 minutes)</td>
<td></td>
</tr>
</tbody>
</table>

### 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

You can verify the connectivity from the controller to the Connected Services public connector server from your location. 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 server (https://rseprod.abb.com (CSE 1.0) or https://cse.abbrobotics.abb.com (CSE 2.0)) 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.
In case of CSE 2.0, for connectivity to ABB Ability, the following servers on Microsoft Azure need to be reachable on port 443 (HTTPS/MQTT):

- DPS: global.azure-devices-provisioning.net
- IoTHub: *.azure-devices.net
- BlobStorage: *.blob.core.windows.net

**Note**

Connected Services Gateway

For more details, see the section ABB Ability™ 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.

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 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.
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:

<table>
<thead>
<tr>
<th>Log number</th>
<th>Log name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>BASE_MODE_UNKNOWN</td>
<td>Controller received unsupported reset command from external source.</td>
</tr>
<tr>
<td>3170</td>
<td>BASE_MODE_FAIL_UPDATE</td>
<td>IRC5 Compatibility Data collector update has been failed.</td>
</tr>
<tr>
<td>8000</td>
<td>GLOBAL_INIT_OK</td>
<td>Connected Services Embedded (CSE) is successfully registered with CMSS server.</td>
</tr>
<tr>
<td>8033</td>
<td>INFO_JAVA_SPECIFIC_LOAD_NOK</td>
<td>Connected Services Embedded (CSE) is unable to start Data collector.</td>
</tr>
<tr>
<td>8123</td>
<td>INFO_EXT_JAVARESET</td>
<td>Connected Services Embedded (CSE) received reset command from external source.</td>
</tr>
</tbody>
</table>
### 8210  INFO_STAY_ALIVE

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.

**Example:**

- **message_count** Alive Bytes:xx/yy
- **HTTP:** p/q/r/s/t/u/v
- **Mem:** 7069164 Run:746 RC:0 LHE:
  - 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.

---

**Note**

**INFO_STAY_ALIVE** is a periodic log.

---

### 8213  INFO_ALIVE_STARTED

Connected Services Embedded (CSE) will send keep alive message once CSE registered. The message format is same as 8210 INFO_STAY_ALIVE.

### 8214  INFO_ALIVE_ENDED

Connected Services Embedded (CSE) will send this alive message when Connected Services Embedded stopped. The message format is same as 8210 INFO_STAY_ALIVE.

### 8700  INFO_MODULE_UPDATE_OK

Connected Services Embedded (CSE) module updated successfully.

### 8701  INFO_MODULE_UPDATE_NOK

Connected Services Embedded (CSE) module update failed.

### 8702  INFO_MODULE_UPDATE_ERROR

Connected Services Embedded (CSE) module update is failed.

### 8801  INFO_S24_STARTED

Connected Services Embedded (CSE) data collector module start.

### 8803  INFO_S2301_STARTED

Connected Services Embedded (CSE) Ability connect- or(S2301) connector started.

### 9003  INFO_SPECIFIC_STOP

Connected Services Embedded (CSE) data collector stopped.
2.3.11 Network topology scenarios

**Connection Type – Ability 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 Ability Connection Type, which means all the communication to the ABB Ability™ Cloud will pass and be routed through the Connected Services Gateway 3G module.

---

**ABB Ability™ Controller 2**

**Controller 1**

**Network Switch**

---

### 3G connection settings for controller 1 and 2

<table>
<thead>
<tr>
<th>State</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>APN (Access point)</td>
<td>abbrobotics.com</td>
</tr>
<tr>
<td>Operator</td>
<td>Automatic</td>
</tr>
<tr>
<td>Band</td>
<td>Automatic</td>
</tr>
<tr>
<td>Auth</td>
<td>Automatic</td>
</tr>
<tr>
<td>Roaming</td>
<td>Enabled</td>
</tr>
<tr>
<td>Idle</td>
<td>0</td>
</tr>
<tr>
<td>Delay</td>
<td>0</td>
</tr>
</tbody>
</table>

### Connected services settings

<table>
<thead>
<tr>
<th>State</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection type</td>
<td>Ability</td>
</tr>
<tr>
<td>Proxy used</td>
<td>No</td>
</tr>
</tbody>
</table>
2.3.11 Network topology scenarios

<table>
<thead>
<tr>
<th>Connected services settings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Server polling</td>
<td>Slow or Fast</td>
</tr>
<tr>
<td>Connected services mode</td>
<td>2.0 Omnicore [Preview]</td>
</tr>
<tr>
<td></td>
<td>1.0 IRC5 compatibility</td>
</tr>
</tbody>
</table>

Continued on next page
Connection Type – Ability 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 Ability™ 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 Ability™ Cloud.

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

### Wi-Fi connection settings for controller 1 and 2

<table>
<thead>
<tr>
<th>State</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSID</td>
<td>SSID-123</td>
</tr>
<tr>
<td>Key</td>
<td>1234567890</td>
</tr>
<tr>
<td>Security</td>
<td>Automatic</td>
</tr>
</tbody>
</table>

### Connected services settings

<table>
<thead>
<tr>
<th>State</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection type</td>
<td>Ability</td>
</tr>
<tr>
<td>Proxy used</td>
<td>Yes or No</td>
</tr>
<tr>
<td>Server polling</td>
<td>Fast or Slow</td>
</tr>
<tr>
<td>Connected services mode</td>
<td>2.0 Omnicore [Preview]</td>
</tr>
<tr>
<td></td>
<td>1.0 IRC5 compatibility</td>
</tr>
</tbody>
</table>
Connection Type – Ability 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 Ability™ 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 Connected Services Gateway Wired Settings table for a simple network configuration.

Connected Services is configured with Ability Connection Type, which means all the communication to the ABB Ability™ Cloud will pass and be routed through the Connected Services Gateway Wired module and not on the public network.
### 2.3.11 Network topology scenarios

Continued

<table>
<thead>
<tr>
<th>Default gateway</th>
<th>Controller 1</th>
<th>Controller 2</th>
<th>Internet Gateway</th>
<th>DHCP server</th>
<th>DNS server</th>
<th>User PC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>172.16.16.1</td>
<td>172.16.16.1</td>
<td>172.16.16.1</td>
<td>172.16.16.1</td>
<td>172.16.16.1</td>
<td></td>
</tr>
</tbody>
</table>

**Connected services settings**

<table>
<thead>
<tr>
<th>State</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection type</td>
<td>Ability</td>
</tr>
<tr>
<td>Proxy used</td>
<td>No</td>
</tr>
<tr>
<td>Server polling</td>
<td>Fast</td>
</tr>
<tr>
<td>Connected services mode</td>
<td>2.0 Omnicore [Preview]</td>
</tr>
<tr>
<td></td>
<td>1.0 IRC5 compatibility</td>
</tr>
</tbody>
</table>
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 as shown in the following figure.
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 as shown in the following figure:

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 Ability.
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 76.

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.
   The following figure and table show a typical network infrastructure:

<table>
<thead>
<tr>
<th>DHCP configuration</th>
<th>Controller 1</th>
<th>Controller 2</th>
<th>Controller 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>172.16.16.58</td>
<td>172.16.16.59</td>
<td>172.16.16.60</td>
</tr>
<tr>
<td>Mask</td>
<td>255.255.255.0</td>
<td>255.255.255.0</td>
<td>255.255.255.0</td>
</tr>
</tbody>
</table>

Continues on next page
Controller with DHCP and manual DNS

The following procedure provides information about configuring the Connected Services from the FlexPendant for controller with DHCP and manual DNS.

<table>
<thead>
<tr>
<th>Action</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the ABB menu, select Control Panel.</td>
</tr>
<tr>
<td>2</td>
<td>Select Configuration.</td>
</tr>
<tr>
<td>3</td>
<td>From Topics, select Communication.</td>
</tr>
<tr>
<td>4</td>
<td>Select IP Route and tap Add.</td>
</tr>
</tbody>
</table>
| 5 | Enter the details for Destination, Gateway, and Label.  
• If DNS IP is entered manually, add the routing for the DNS IP.  
• In this example, Destination: 8.8.8.8/31 is Google DNS. |
| 6 | Tap OK and restart the controller to take effect of the changes. |

Steps to configure DNS manually

<table>
<thead>
<tr>
<th>Action</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the ABB menu, select Control Panel.</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Action</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Select Configuration.</td>
</tr>
<tr>
<td>3</td>
<td>From Topics, select Communication.</td>
</tr>
<tr>
<td>4</td>
<td>Select IP Route and tap Add.</td>
</tr>
</tbody>
</table>
| 5      | Enter the details for Destination, Gateway, and Label.  
• Enter the Gateway IP as box IP. In this example, it is 172.16.16.25. |
| 6      | Tap OK and restart the controller to take effect of the changes. |

**Note**

Manually define the DNS, if it is not provided automatically. Also, define a route to go through the gateway box for the DNS IP.
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.

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

Continues on next page
Configuration

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoveAtPpToMain</td>
<td>It is possible to configure if the cyclically evaluated logical conditions</td>
</tr>
<tr>
<td></td>
<td>shall be removed or not when setting the program pointer to <code>main</code>.</td>
</tr>
<tr>
<td></td>
<td>- <strong>On</strong> - remove.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Off</strong> - do not remove (default behavior).</td>
</tr>
<tr>
<td>ErrorMode</td>
<td>It is possible to configure which error mode to use when the evaluation of</td>
</tr>
<tr>
<td></td>
<td>a Cyclic bool fails.</td>
</tr>
<tr>
<td></td>
<td>- <strong>SysStopError</strong>(^{i}) - stop RAPID execution and produce an error</td>
</tr>
<tr>
<td></td>
<td>log (default behavior).</td>
</tr>
<tr>
<td></td>
<td>- <strong>Warning</strong> - produce a warning log.</td>
</tr>
<tr>
<td></td>
<td>- <strong>None</strong> - do nothing.</td>
</tr>
<tr>
<td>RecoveryMode</td>
<td>It is possible to configure if a failing Cyclic bool shall be recovered</td>
</tr>
<tr>
<td></td>
<td>or not.</td>
</tr>
<tr>
<td></td>
<td>- <strong>On</strong> - try to recover the evaluation of a failing Cyclic bool (default</td>
</tr>
<tr>
<td></td>
<td>behavior).</td>
</tr>
<tr>
<td></td>
<td>- <strong>Off</strong> - do not try to recover the evaluation of a Cyclic bool.</td>
</tr>
</tbody>
</table>

\(^{i}\) 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: `signaldi`, `signaldo` or physical `di` and `do`
  - Object type: `VAR`
- Operands: `NOT`, `AND`, `OR`, `XOR`, `=`, `'(', ')'`

\Signal shall be of:
- Object type: `signaldo`

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.
• 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

```plaintext
! 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

```plaintext
! 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

```plaintext
! 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=1000000;
    SetupCyclicBool cyclicbool2, num1=8 OR dnum1=11000000;
    WaitUntil cyclicbool1=TRUE;
    ...
    WaitUntil cyclicbool2=TRUE;
    ...
    ! Remove all connections when no longer in use
    RemoveAllCyclicBool;
ENDPROC
```
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
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.

```plaintext
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
```

Continued
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>• RemoveAtPpToMain</td>
</tr>
<tr>
<td></td>
<td>• ErrorMode</td>
</tr>
<tr>
<td>RemoveAtPpToMain</td>
<td>The action value RemoveAtPpToMain is used to configure if a connected Cyclic bool shall be removed or not when setting the program pointer to Main.</td>
</tr>
<tr>
<td>ErrorMode</td>
<td>The action value ErrorMode is used to configure which error mode to use when evaluation fails.</td>
</tr>
<tr>
<td>RecoveryMode</td>
<td>The action value RecoveryMode is used to configure which recovery mode to use when evaluation fails.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetupCyclicBool</td>
<td>SetupCyclicBool connects a logical condition to a boolean variable.</td>
</tr>
<tr>
<td>RemoveCyclicBool</td>
<td>RemoveCyclicBool removes a specific connected logical condition.</td>
</tr>
<tr>
<td>RemoveAllCyclicBool</td>
<td>RemoveAllCyclicBool removes all connected logical conditions.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetMaxNumberOfCyclicBool</td>
<td>GetMaxNumberOfCyclicBool retrieves the maximum number of cyclically evaluated logical condition that can be connected at the same time.</td>
</tr>
<tr>
<td>GetNextCyclicBool</td>
<td>GetNextCyclicBool retrieves the name of a connected cyclically evaluated logical condition.</td>
</tr>
<tr>
<td>GetNumberOfCyclicBool</td>
<td>GetNumberOfCyclicBool retrieves the number of a connected cyclically evaluated logical condition.</td>
</tr>
<tr>
<td>IsCyclicBool</td>
<td>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.</td>
</tr>
</tbody>
</table>

Data types

*Cyclic bool* includes no data types.
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 121.

**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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PackDNHeader</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 RobotWare-OS

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 121.

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
2.6 File and I/O device handling

2.6.1 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:

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

2.6.2.1 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 119.

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.
2.6.2.2 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*.

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

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*.

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

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*.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadNum</td>
<td><em>ReadNum</em> is used to read a number from a character based file or I/O device.</td>
</tr>
<tr>
<td>ReadStr</td>
<td><em>ReadStr</em> is used to read a string from a character based file or I/O device.</td>
</tr>
<tr>
<td>ReadBin</td>
<td><em>ReadBin</em> 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.</td>
</tr>
<tr>
<td>ReadStrBin</td>
<td><em>ReadStrBin</em> is used to read a string from a binary I/O device or file.</td>
</tr>
</tbody>
</table>
2.6.2.3 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 enq
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
! 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
2.6.3 Raw data communication

2.6.3.1 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 123.

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.
2.6.3.2 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.

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

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.

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

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RawBytesLen</td>
<td>RawBytesLen is used to get the valid data length in a rawbytes variable.</td>
</tr>
</tbody>
</table>
2.6.3.3 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.

```plaintext
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
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);}
2.6.4 File and directory management

2.6.4.1 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 127*.

1. Open a directory.
2. Read from the directory and search until you find what you are looking for.
3. Close the directory.
2.6.4.2 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.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir</td>
<td>dir contains a reference to a directory on disk or network. It can be linked to the physical directory with the instruction OpenDir.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenDir</td>
<td>OpenDir is used to open a directory.</td>
</tr>
<tr>
<td>CloseDir</td>
<td>CloseDir is used to close a directory.</td>
</tr>
<tr>
<td>MakeDir</td>
<td>MakeDir is used to create a new directory.</td>
</tr>
<tr>
<td>RemoveDir</td>
<td>RemoveDir is used to remove an empty directory.</td>
</tr>
<tr>
<td>CopyFile</td>
<td>CopyFile is used to make a copy of an existing file.</td>
</tr>
<tr>
<td>RenameFile</td>
<td>RenameFile 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.</td>
</tr>
<tr>
<td>RemoveFile</td>
<td>RemoveFile is used to remove a file.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadDir</td>
<td>ReadDir is used to retrieve the name of the next file or subdirectory under a directory that has been opened with the instruction OpenDir. Note that the first items read by ReadDir are . (full stop character) and .. (double full stop characters) symbolizing the current directory and its parent directory.</td>
</tr>
<tr>
<td>FileSize</td>
<td>FileSize is used to retrieve the size (in bytes) of the specified file.</td>
</tr>
<tr>
<td>FSSize</td>
<td>FSSize (File System Size) is used to retrieve the size (in bytes) of the file system in which a specified file resides. FSSize can either retrieve the total size or the free size of the system.</td>
</tr>
<tr>
<td>IsFile</td>
<td>IsFile test if the specified file is of the specified type. It can also be used to test if the file exist at all.</td>
</tr>
</tbody>
</table>
2.6.4.3 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 lsdir(string dirname)
VAR dir directory;
VAR string filename;
!
CHECK that dirname really is a directory
IF IsFile(dirname \\Directory) THEN
  ! Open the directory
  OpenDir directory, dirname;
!
LOOP through the files in the directory
WHILE ReadDir(directory, filename) DO
  IF (filename <> "." AND filename <> ".." THEN
    TPWrite filename;
  ENDIF
ENDWHILE
!
CLOSE the directory
CloseDir directory;
ENDIF
ENDP

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
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.

```plaintext
VAR num freesyssize;
VAR num f_size;

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

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

! Copy file if enough space free
IF f_size < freesyssize THEN
  CopyFile "HOME:/myfile", "HOME:/yourfile";
ENDIF
```
2.7 Fixed Position Events

2.7.1 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 133.

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.
2.7.2 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.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>triggdata</td>
<td><em>triggdata</em> is used to store data about a position event.\nA 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.\ntriggdata also contains information on when the action shall occur, for example when the TCP is at a defined distance from the target.\ntriggdata is a non-value data type.</td>
</tr>
<tr>
<td>triggios</td>
<td><em>triggios</em> is used to store data about a position event used by the instruction <em>TriggLIOs</em>.\ntriggios sets the value of an output signal using a <em>num</em> value.</td>
</tr>
<tr>
<td>triggiosdnum</td>
<td><em>triggiosdnum</em> is used to store data about a position event used by the instruction <em>TriggLIOs</em>.\ntriggiosdnum sets the value of an output signal using a <em>dnum</em> value.</td>
</tr>
<tr>
<td>triggstrgo</td>
<td><em>triggstrgo</em> is used to store data about a position event used by the instruction <em>TriggLIOs</em>.\ntriggstrgo sets the value of an output signal using a <em>stringdig</em> value (string containing a number).</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TriggIO</td>
<td><em>TriggIO</em> defines the setting of an output signal and when to set that signal. The definition is stored in a variable of type <em>triggdata</em>.\nTriggIO 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.\nBy 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).</td>
</tr>
<tr>
<td>TriggEquip</td>
<td><em>TriggEquip</em> works like <em>TriggIO</em>, with the difference that <em>TriggEquip</em> can compensate for the internal delay of the external equipment.\nFor example, the signal to a glue gun must be set a short time before the glue is pressed out and the gluing begins.</td>
</tr>
<tr>
<td>TriggInt</td>
<td><em>TriggInt</em> defines when to run an interrupt routine. The definition is stored in a variable of type <em>triggdata</em>.\n<em>TriggInt</em> 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).</td>
</tr>
</tbody>
</table>
### 2.7.2 RAPID components and system parameters

**Instruction** | **Description**
--- | ---
TriggCheckIO | 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 triggdata.
| 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.
| 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.
| 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).

TriggRampAO | 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 triggdata.
| TriggRampAO defines where the ramping of the signal is to start and the length of the ramping.

TriggL | TriggL is a move instruction, similar to MoveL. In addition to the movement the TriggL instruction can set output signals, run interrupt routines and check input or output signals at fixed positions.
| TriggL executes up to 8 position events stored as triggdata. These must be defined before calling TriggL.

TriggC | TriggC is a move instruction, similar to MoveC. In addition to the movement the TriggC instruction can set output signals, run interrupt routines and check input or output signals at fixed positions.
| TriggC executes up to 8 position events stored as triggdata. These must be defined before calling TriggC.

TriggJ | TriggJ is a move instruction, similar to MoveJ. In addition to the movement the TriggJ instruction can set output signals, run interrupt routines and check input or output signals at fixed positions.
| TriggJ executes up to 8 position events stored as triggdata. These must be defined before calling TriggJ.

TriggLIOs | TriggLIOs is a move instruction, similar to MoveL. In addition to the movement the TriggLIOs instruction can set output signals at fixed positions.
| TriggLIOs is similar to the combination of TriggEquip and TriggL. The difference is that TriggLIOs can handle up to 50 position events stored as an array of datatype triggios, triggiosdnum, or triggstrgo.

MoveLSync | MoveLSync is a linear move instruction that calls a procedure in the middle of the corner path.

MoveCSync | MoveCSync is a circular move instruction that calls a procedure in the middle of the corner path.

MoveJSync | MoveJSync is a joint move instruction that calls a procedure in the middle of the corner path.

**Functions**

Fixed Position Events includes no RAPID functions.

---

Continues on next page
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Preset Time</td>
<td>TriggEquip 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 Event preset time. Event preset time belongs to the type Motion System in the topic Motion.</td>
</tr>
</tbody>
</table>
2.7.3 Code examples

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

\[
\begin{align*}
\text{MoveJ } & p1, \text{ vmax, fine, tool1;} \\
\text{MoveL } & p2, \text{ v1000, z20, tool1;} \\
\text{SetDO } & \text{ do1, 1;} \\
\text{MoveL } & p3, \text{ v1000, fine, tool1;} \\
\end{align*}
\]

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

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.

\[
\begin{align*}
\text{VAR } & \text{triggdata do_set;} \\
! \text{Define that do1 shall be set when 30 mm from target} \\
\text{TriggIO } & \text{do_set, 30 \ DOp:=do1, 1;} \\
\text{MoveJ } & p1, \text{ vmax, fine, tool1;} \\
! \text{Move to p2 and let system execute do_set} \\
\text{TriggL } & p2, \text{ v1000, do_set, z20, tool1;} \\
\text{MoveL } & p3, \text{ v1000, fine, tool1;} \\
\end{align*}
\]
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).

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.

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

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

2.8.1 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.
## 2.8.2 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 136*.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Specifies the name of the cross connection.</td>
</tr>
<tr>
<td><strong>Resultant</strong></td>
<td>The I/O signal that receive the result of the cross connection as its new value.</td>
</tr>
<tr>
<td><strong>Actor 1</strong></td>
<td>The first I/O signal to be used in the evaluation of the Resultant.</td>
</tr>
<tr>
<td><strong>Invert actor 1</strong></td>
<td>If <em>Invert actor 1</em> is set to Yes, then the inverted value of <em>Actor 1</em> is used in the evaluation of the Resultant.</td>
</tr>
<tr>
<td><strong>Operator 1</strong></td>
<td>Operand between <em>Actor 1</em> and <em>Actor 2</em>. Can be either of the operands:</td>
</tr>
<tr>
<td></td>
<td>• <strong>AND</strong> - Results in the value 1 if both input values are 1.</td>
</tr>
<tr>
<td></td>
<td>• <strong>OR</strong> - Results in the value 1 if at least one of the input values are 1.</td>
</tr>
</tbody>
</table>

**Note**

The operators are calculated left to right (*Operator 1* first and *Operator 4* last).

| **Actor 2** | The second I/O signal (if more than one) to be used in the evaluation of the Resultant. |
| **Invert actor 2** | If *Invert actor 2* is set to Yes, then the inverted value of *Actor 2* is used in the evaluation of the Resultant. |
| **Operator 2** | Operand between *Actor 2* and *Actor 3*.  |
| | See *Operator 1*.  |
| **Actor 3** | The third I/O signal (if more than two) to be used in the evaluation of the Resultant. |
| **Invert actor 3** | If *Invert actor 3* is set to Yes, then the inverted value of *Actor 3* is used in the evaluation of the Resultant. |
| **Operator 3** | Operand between *Actor 3* and *Actor 4*.  |
| | See *Operator 1*.  |
| **Actor 4** | The fourth I/O signal (if more than three) to be used in the evaluation of the Resultant. |
| **Invert actor 4** | If *Invert actor 4* is set to Yes, then the inverted value of *Actor 4* is used in the evaluation of the Resultant. |
| **Operator 4** | Operand between *Actor 4* and *Actor 5*.  |
| | See *Operator 1*.  |
| **Actor 5** | The fifth I/O signal (if all five are used) to be used in the evaluation of the Resultant. |
| **Invert actor 5** | If *Invert actor 5* is set to Yes, then the inverted value of *Actor 5* is used in the evaluation of the Resultant. |
2.8.3 Examples

Logical AND

The following logical structure...

\[ \text{di1} \quad \text{do1} \quad \& \quad \text{do2} \]

\[ \text{xx0300000457} \]

... is created as shown below.

<table>
<thead>
<tr>
<th>Resultant</th>
<th>Actor 1</th>
<th>Invert actor 1</th>
<th>Operator 1</th>
<th>Actor 2</th>
<th>Invert actor 2</th>
<th>Operator 2</th>
<th>Actor 3</th>
<th>Invert actor 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>do26</td>
<td>di1</td>
<td>No</td>
<td>AND</td>
<td>do2</td>
<td>No</td>
<td>AND</td>
<td>do10</td>
<td>No</td>
</tr>
</tbody>
</table>

Logical OR

The following logical structure...

\[ \text{di1} \quad \text{do2} \quad \text{do10} \]

\[ \text{xx0300000459} \]

... is created as shown below.

<table>
<thead>
<tr>
<th>Resultant</th>
<th>Actor 1</th>
<th>Invert actor 1</th>
<th>Operator 1</th>
<th>Actor 2</th>
<th>Invert actor 2</th>
<th>Operator 2</th>
<th>Actor 3</th>
<th>Invert actor 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>do26</td>
<td>di1</td>
<td>No</td>
<td>OR</td>
<td>do2</td>
<td>No</td>
<td>OR</td>
<td>do10</td>
<td>No</td>
</tr>
</tbody>
</table>

Inverted signals

The following logical structure (where a ring symbolize an inverted signal)...
... but with three cross connections it can be implemented as shown below.

<table>
<thead>
<tr>
<th>Resultant</th>
<th>Actor 1</th>
<th>Invert actor 1</th>
<th>Operator 1</th>
<th>Actor 2</th>
<th>Invert actor 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>di17</td>
<td>di1</td>
<td>No</td>
<td>AND</td>
<td>do2</td>
<td>No</td>
</tr>
<tr>
<td>do26</td>
<td>di1</td>
<td>No</td>
<td>AND</td>
<td>do2</td>
<td>No</td>
</tr>
<tr>
<td>do13</td>
<td>di1</td>
<td>No</td>
<td>AND</td>
<td>do2</td>
<td>No</td>
</tr>
</tbody>
</table>

Complex conditions

The following logical structure...

... is created as shown below.

<table>
<thead>
<tr>
<th>Resultant</th>
<th>Actor 1</th>
<th>Invert actor 1</th>
<th>Operator 1</th>
<th>Actor 2</th>
<th>Invert actor 2</th>
<th>Operator 2</th>
<th>Actor 3</th>
<th>Invert actor 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>do11</td>
<td>di2</td>
<td>No</td>
<td>AND</td>
<td>do3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do14</td>
<td>di12</td>
<td>No</td>
<td>AND</td>
<td>do3</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>di11</td>
<td>di13</td>
<td>No</td>
<td>AND</td>
<td>do3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do23</td>
<td>di13</td>
<td>No</td>
<td>AND</td>
<td>do3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do17</td>
<td>di13</td>
<td>No</td>
<td>AND</td>
<td>do3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do15</td>
<td>do11</td>
<td>No</td>
<td>OR</td>
<td>do14</td>
<td>No</td>
<td>OR</td>
<td>di11</td>
<td>Yes</td>
</tr>
<tr>
<td>do33</td>
<td>di11</td>
<td>No</td>
<td>AND</td>
<td>do23</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do61</td>
<td>do17</td>
<td>No</td>
<td>AND</td>
<td>do3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>do54</td>
<td>do15</td>
<td>No</td>
<td>OR</td>
<td>do33</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.8.4 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 RAPID Message Queue

2.9 RAPID Message Queue

2.9.1 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 147.

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.
2.9.2 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.

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
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:**

```
"robtarget;[[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.
2 RobotWare-OS

2.9.2 RAPID Message Queue behavior

Related information

For more information on queues and messages, see Technical reference manual - RAPID kernel.
2.9.3 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMQ Type</td>
<td>Can have one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• None - Disable all communication with RAPID Message Queue for this RAPID task.</td>
</tr>
<tr>
<td></td>
<td>• Internal - 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.</td>
</tr>
<tr>
<td></td>
<td>• Remote - Enable communication with RAPID Message Queue for this task, both with other tasks on the controller and external clients (FlexPendant and PC applications). The default value is None.</td>
</tr>
<tr>
<td>RMQ Mode</td>
<td>Defines the mode of the queue.</td>
</tr>
<tr>
<td></td>
<td>Can have one of the following values:</td>
</tr>
<tr>
<td></td>
<td>• Interrupt - A message can only be received by connecting a trap routine to a specified message type.</td>
</tr>
<tr>
<td></td>
<td>• Synchronous - A message can only be received by executing an RMQReadWait instruction.</td>
</tr>
<tr>
<td></td>
<td>Default value is Interrupt.</td>
</tr>
<tr>
<td>RMQ Max Message Size</td>
<td>The maximum data size, in bytes, for a RAPID Message Queue message. An integer between 400 and 3000. The default value is 400.</td>
</tr>
<tr>
<td>Note</td>
<td>The value cannot be changed in RobotStudio or on the FlexPendant. The only way to change the value is to edit the sys.cfg file by adding the attribute RmqMaxMsgSize with the desired value.</td>
</tr>
<tr>
<td>RMQ Max No Of Messages</td>
<td>The maximum number of RAPID Message Queue messages in the queue to this task. An integer between 1 and 10. The default value is 5.</td>
</tr>
<tr>
<td>Note</td>
<td>The value cannot be changed in RobotStudio or on the FlexPendant. The only way to change the value is to edit the sys.cfg file by adding the attribute RmqMaxNoOfMsg with the desired value.</td>
</tr>
</tbody>
</table>
2.9.4 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

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

Functions

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

Data types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmqslot</td>
<td>Slot identity of a RAPID task or Robot Application Builder client.</td>
</tr>
<tr>
<td>rmqmessage</td>
<td>A message used to store data in when communicating with RAPID Message</td>
</tr>
<tr>
<td></td>
<td>Queue. It contains information about what type of data is sent, the slot</td>
</tr>
<tr>
<td></td>
<td>identity of the sender, and the actual data.</td>
</tr>
<tr>
<td>Note: rmqmessage</td>
<td>is a large data type. Declaring too many variables of this data type can</td>
</tr>
<tr>
<td></td>
<td>lead to memory problems. Reuse the same rmqmessage variables as much as</td>
</tr>
<tr>
<td></td>
<td>possible.</td>
</tr>
<tr>
<td>rmqheader</td>
<td>The rmqheader describes the message and can be read by the RAPID program.</td>
</tr>
</tbody>
</table>
2.9.5 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()
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];  
        | 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";
! 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
    ResetRetryCount;
    RETRY;
  ENDIF
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;
      }
    }
  }
}
// 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));
2.10 Socket Messaging

2.10.1 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 156.

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.
2.10.2 Schematic picture of socket communication

Illustration of socket communication

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).
2.10.3 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.

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

For more information, see *Technical reference manual - RAPID kernel*, section *String literals.*
2.10.4 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.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socketdev</td>
<td>A socket device used to communicate with other computers on a network.</td>
</tr>
<tr>
<td>socketstatus</td>
<td>Can contain status information from a socketdev variable.</td>
</tr>
</tbody>
</table>

Instructions for client

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

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

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:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocketBind</td>
<td>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.</td>
</tr>
<tr>
<td>SocketListen</td>
<td>Makes the computer act as a server and accept incoming connections. It will listen for a connection on the port specified by SocketBind.</td>
</tr>
<tr>
<td>SocketAccept</td>
<td>Accepts an incoming connection request. Used by the server to accept the client’s request.</td>
</tr>
</tbody>
</table>
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*.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| `SocketGetStatus` | Returns information about the last instruction performed on the socket (created, connected, bound, listening, closed).  
  `SocketGetStatus` does not detect changes from outside RAPID (such as a broken connection). |
2.10.5 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:

```rapid
! 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):

```rapid
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;
  ENDWHILE
ENDPROC
```

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;
ENDIF
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;
ENDIF
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
2.11 User logs

2.11.1 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:

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

---

**What is included**

Every *Absolute Accuracy* robot is delivered with:

- compensation parameters saved on the robot’s serial measurement board
3 Motion Performance

3.1.1 About Absolute Accuracy

Continued

- 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. Compensation parameters saved in the robot’s serial measurement board differ depending on which Absolute Accuracy option is selected.

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.
3.1.3 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. For a MultiMove system, configure the parameter Use Robot Calibration for each robot. It should be set to r2_calib for robot 2, r3_calib for robot 3, and r4_calib for robot 4.
6. Restart the controller for the changes to take effect.

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. For a MultiMove system, repeat step 3 and 4 for each robot. Use Robot Calibration is then set to “r2_uncalib” for robot 2, “r3_uncalib” for robot 3 and “r4_uncalib” for robot 4.
6. Restart the controller for the changes to take effect.

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’s serial measurement board 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 Motion Performance

3.1.4.1 Maintenance that affect the accuracy

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 178.

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 164.

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 3D measurement system.

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

Continues on next page
A summary of the calibration process is presented as follows:

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

**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 162*. 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 176*. 

3 Motion Performance

3.1.4.1 Maintenance that affect the accuracy

Continued
## 3 Motion Performance

### 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.

<table>
<thead>
<tr>
<th>If...</th>
<th>...then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the tool is not properly calibrated</td>
<td>recalibrate if the TCP has changed.</td>
</tr>
<tr>
<td>the tool load is not correctly defined</td>
<td>run Load Identification to ensure correct mass, centre of gravity and inertia for the active tool.</td>
</tr>
<tr>
<td>the resolver offsets are no longer valid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Check that the axis scales show that the robot stands correctly in the home position.</td>
</tr>
<tr>
<td></td>
<td>2 If the indicators are not aligned, move the robot to correct position and update the revolution counters.</td>
</tr>
<tr>
<td></td>
<td>3 If the indicators are close to aligned but not correct, re-calibrate with the standard calibration for the robot.</td>
</tr>
<tr>
<td>the robot’s relationship to the fixture(s) has changed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Check by moving the robot to a predefined position on the fixture(s).</td>
</tr>
<tr>
<td></td>
<td>2 Visually assessing whether the deviation is excessive.</td>
</tr>
<tr>
<td></td>
<td>3 If excessive, realign robot to fixture(s).</td>
</tr>
<tr>
<td>the robot structure has changed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Visually assess whether the robot is damaged.</td>
</tr>
<tr>
<td></td>
<td>2 If damaged then replace entire manipulator -or- replace affected arm(s) -or- recalibrate affected arm(s).</td>
</tr>
</tbody>
</table>
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.
3 Motion Performance

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.

![Desired position](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.

![Position due to deflection](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.

![Fake target](xx0300000226)
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.
3 Motion Performance

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 on the serial measurement board
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 joint target 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
Absolute Accuracy verification

The parameters are loaded onto the controller and activated. The robot is then run to a set of 50 robtarget 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 on the robot's serial measurement board (see Compensation parameters on page 173).

A birth certificate is created representing the Absolute Accuracy measurement protocol for the calibration and verification sequence (see Birth certificate on page 172).
3 Motion Performance

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)
3.1.6.3 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 on the robot’s serial measurement board.

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 162.

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 Installation Manager. 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 Motion Performance

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:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Measure fixture alignment</td>
</tr>
<tr>
<td>2</td>
<td>Measure robot alignment</td>
</tr>
<tr>
<td>3</td>
<td>Calculate frame relationships</td>
</tr>
<tr>
<td>4</td>
<td>Calibrate tool</td>
</tr>
</tbody>
</table>

Illustration
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

<table>
<thead>
<tr>
<th>Measurement positions</th>
<th>Reference positions</th>
<th>Frame relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos1: 100, 100, 200</td>
<td>Pos1: 100, 100, 100</td>
<td>1) RobotStudio work object (0,0,-100,0,0,0) (x,y,z,roll,pitch,yaw)</td>
</tr>
<tr>
<td>Pos2: 100, 200, 200</td>
<td>Pos2: 100, 200, 100</td>
<td></td>
</tr>
<tr>
<td>Pos3: 200, 200, 200</td>
<td>Pos3: 200, 200, 100</td>
<td></td>
</tr>
<tr>
<td>Pos4: 200, 100, 200</td>
<td>Pos4: 200, 100, 100</td>
<td></td>
</tr>
</tbody>
</table>
3 Motion Performance

3.1.7.3 Measure robot alignment

## 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:

<table>
<thead>
<tr>
<th>Alignment procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment to physical base</td>
<td>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.</td>
</tr>
<tr>
<td>Alignment to theoretical base</td>
<td>Measuring several robot poses and letting the alignment software determine the robot alignment.</td>
</tr>
</tbody>
</table>

### 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 have 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.
3.1.7.4 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 Motion Performance

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.
3.2 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 180.
- Changing Motion Process Mode from RAPID, see Motion Process Mode [included in 3100-1] on page 188.
- Wrist Move, see Wrist Move [included in 3100-1] on page 196.
3.3 Advanced Shape Tuning [included in 3100-1]

3.3.1 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 185.
2. Perform automatic tuning of the joint friction levels using the instructions FricIdInit and FricIdEvaluate. See Automatic friction tuning on page 181.
3. Compensate for the friction using the instruction FricIdSetFricLevels.
3.3.2 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 186.
- 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.
3 Motion Performance

3.3.2 Automatic friction tuning

Continued

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.

```plaintext
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.
3.3.3 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:

<table>
<thead>
<tr>
<th>Tune type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TUNE_FRIC_LEV</td>
<td>By calling the instruction TuneServo 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 <em>Friction FFW Level</em>.</td>
</tr>
<tr>
<td>TUNE_FRIC_RAMP</td>
<td>By calling the instruction TuneServo 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 <em>Friction FFW Ramp</em>. There is normally no need to tune the friction ramp.</td>
</tr>
</tbody>
</table>

Configure friction level

The friction level is set for each robot joint. Perform the following steps for one joint at a time:

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Continues on next page
3 Motion Performance

3.3.3 Manual friction tuning

Continued

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4</strong> The final tuning values can be transferred to the system parameters. Example: The <em>Friction FFW Level</em> is 0.5 and the final tune value (<em>TUNE_FRIC_LEV</em>) is 120%. Set <em>Friction FFW Level</em> to 0.6 and tune value to 100% (default value), which is equivalent.</td>
</tr>
</tbody>
</table>

**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 186.
3.3.4 System parameters

3.3.4.1 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friction FFW On</td>
<td>Advanced Shape Tuning is active when Friction FFW On is set to TRUE.</td>
</tr>
<tr>
<td>Friction FFW Level</td>
<td>Friction FFW Level is the friction level for the robot joint. See illustration below.</td>
</tr>
<tr>
<td>Friction FFW Ramp</td>
<td>Friction FFW Ramp is the speed of the robot motor shaft, at which the friction has reached the friction level defined by Friction FFW Level. See illustration below. There is normally no need to make changes to Friction FFW Ramp.</td>
</tr>
</tbody>
</table>

Illustration

![Illustration of Friction FFW parameters](en0900000117)
### 3.3.4.2 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 186.*

**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:

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong> In RobotStudio, open the Configuration Editor, Motion topic, and select the type (\text{Friction comp (except for the robots IRB 1400 and IRB 1410 where they belong to the type \text{Control parameters}).})</td>
</tr>
<tr>
<td><strong>2</strong> Multiply <em>Friction FFW Level</em> with the tuning value. Set this value as the new <em>Friction FFW Level</em> and set the tuning value ((TUNE_FRIC_LEV)) to 100%. Example: The <em>Friction FFW Level</em> is 0.5 and the final tune value ((TUNE_FRIC_LEV)) is 120%. Set <em>Friction FFW Level</em> to 0.6 (1.20\times0.5) and the tuning value to 100% (default value), which is equivalent.</td>
</tr>
<tr>
<td><strong>3</strong> Restart the controller for the changes to take effect.</td>
</tr>
</tbody>
</table>

**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 an new estimated value.

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong> In RobotStudio, open the Configuration Editor, Motion topic, and select the type (\text{Friction comp (except for the robots IRB 1400 and IRB 1410 where they belong to the type \text{Control parameters}).})</td>
</tr>
<tr>
<td><strong>2</strong> Set the parameter <em>Friction FFW Level</em> to an estimated value. Do not set the value 0 (zero), because that will make tuning impossible.</td>
</tr>
<tr>
<td><strong>3</strong> Restart the controller for the changes to take effect.</td>
</tr>
</tbody>
</table>
3.3.5 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

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FricIdInit</td>
<td>Initiate friction identification</td>
</tr>
<tr>
<td>FricIdEvaluate</td>
<td>Evaluate friction identification</td>
</tr>
<tr>
<td>FricIdSetFricLevels</td>
<td>Set friction levels after friction identification</td>
</tr>
</tbody>
</table>

Functions

Advanced Shape Tuning includes no functions.

Data types

Advanced Shape Tuning includes no data types.
3.4 Motion Process Mode [included in 3100-1]

3.4.1 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.

**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. This is the recommended choice for improving path accuracy on small and medium size robots, for example IRB 2400 and IRB 2600.
- **Low speed accuracy mode** – this mode improves path accuracy. The cycle time will be slightly increased compared to Accuracy mode. This is the recommended choice for improving path accuracy on large size robots, for example IRB 4600.
- **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.

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 *, vmax, ...;
...

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

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
3 Motion Performance

3.4.2 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
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.
3.4.3 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 192*.

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 195*.
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.
3.4.4 Additional information

Motion Process Mode compared to TuneServo and AccSet

Motion process modes simplify application specific tuning and make 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

<table>
<thead>
<tr>
<th>For information about</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration of Motion Process Mode parameters.</td>
<td>Technical reference manual - System parameters</td>
</tr>
<tr>
<td>RAPID instructions:</td>
<td>Technical reference manual - RAPID Instructions, Functions and Data types</td>
</tr>
<tr>
<td>• AccSet - Reduces the acceleration</td>
<td></td>
</tr>
<tr>
<td>• MotionProcessModeSet - Set motion process mode</td>
<td></td>
</tr>
<tr>
<td>• TuneServo - Tuning servos</td>
<td></td>
</tr>
</tbody>
</table>
3 Motion Performance

3.5 Wrist Move [included in 3100-1]

3.5.1 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
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.
3.5.2 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 \texttt{MoveC} instruction. The frame is defined to be equal to the tool frame at the starting position. Note that for a sequence of \texttt{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.

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
Tip

Wrist movement is not limited to circular arcs only: If the targets of \texttt{MoveC} are collinear, then a straight line will be achieved.
### 3.5.3 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*.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CirPathMode</td>
<td><em>CirPathMode</em> makes it possible to select different modes to reorientate the tool during circular movements. The arguments <em>Wrist45, Wrist46, and Wrist56</em> are used specifically for the Wrist Move option.</td>
</tr>
</tbody>
</table>
3.5.4 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 202. 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
3.5.5 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.
This page is intentionally left blank
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 209.

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
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 \texttt{WZLimJointDef} or \texttt{WZHomeJointDef}).

A variable of type \texttt{wzstationary} or \texttt{wztemporary} can not be redefined. They can only be defined once (with \texttt{WZLimSup} or \texttt{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.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wztemporary</td>
<td>wztemporary is used to identify a temporary world zone and can be used anywhere in the RAPID program. 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.</td>
</tr>
<tr>
<td>wzstationary</td>
<td>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 Operating manual - OmniCore. 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. Stationary world zones shall be used if security is involved.</td>
</tr>
<tr>
<td>shapedata</td>
<td>shapedata is used to describe the geometry of a world zone. World zones can be defined in 4 different geometrical shapes: • 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</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WZBoxDef</td>
<td>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 shapedata. The volume can also be defined as the inverse of the box (all volume outside the box).</td>
</tr>
<tr>
<td>WZCylDef</td>
<td>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 shapedata. The volume can also be defined as the inverse of the cylinder (all volume outside the cylinder).</td>
</tr>
<tr>
<td>WZSphDef</td>
<td>WZSphDef is used to define a volume that has the shape of a sphere. The definition is stored in a variable of type shapedata. The volume can also be defined as the inverse of the sphere (all volume outside the sphere).</td>
</tr>
</tbody>
</table>
### 4 Motion Supervision

#### 4.1.2 RAPID components

Continued

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WZLimJointDef</td>
<td>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. 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. The definition is stored in a variable of type shapedata.</td>
</tr>
<tr>
<td>WZHomeJointDef</td>
<td>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. 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. The definition is stored in a variable of type shapedata.</td>
</tr>
<tr>
<td>WZLimSup</td>
<td>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. When calling WZLimSup you specify whether it is a stationary world zone, stored in a wzstationary variable, or a temporary world zone, stored in a wztemporary variable.</td>
</tr>
<tr>
<td>WZDOSet</td>
<td>WZDOSet is used to define, and enable, setting a digital output signal when the TCP reaches the world zone. When calling WZDOSet you specify whether it is a stationary world zone, stored in a wzstationary variable, or a temporary world zone, stored in a wztemporary variable.</td>
</tr>
<tr>
<td>WZDisable</td>
<td>WZDisable is used to disable the supervision of a temporary world zone.</td>
</tr>
<tr>
<td>WZEnable</td>
<td>WZEnable is used to re-enable the supervision of a temporary world zone. A world zone is automatically enabled on creation. Enabling is only necessary after it has been disabled with WZDisable.</td>
</tr>
<tr>
<td>WZFree</td>
<td>WZFree is used to disable and erase a temporary world zone.</td>
</tr>
</tbody>
</table>

### 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*.

```
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.
The shadowed area in the illustration shows the world zone.

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
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.
4.2.3 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

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.

<table>
<thead>
<tr>
<th>When ...</th>
<th>then ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the collision is detected</td>
<td>the motor torques are reversed and the mechanical brakes applied in order to stop the robot</td>
</tr>
<tr>
<td>the robot has stopped</td>
<td>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</td>
</tr>
<tr>
<td>the residual forces are removed</td>
<td>the robot stops again and remains in the motors on state</td>
</tr>
</tbody>
</table>
4 Motion Supervision

4.2.3 What happens at a collision

Continued

Speed and torque diagram

- Motor speed
- Motor torque
- Time of collision detected
- Robot stopped
- Residual forces removed
- Speed reversed
- Torque reversed
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path Collision Detection</td>
<td>Turn the collision detection On or Off for program execution.</td>
</tr>
<tr>
<td>Jog Collision Detection</td>
<td>Turn the collision detection On or Off for jogging.</td>
</tr>
<tr>
<td>Path Collision Detection Level</td>
<td>Modifies the Collision Detection supervision level for program execution by</td>
</tr>
<tr>
<td></td>
<td>the specified percentage value. A large percentage value makes the function</td>
</tr>
<tr>
<td></td>
<td>less sensitive.</td>
</tr>
<tr>
<td>Jog Collision Detection Level</td>
<td>Modifies the Collision Detection supervision level for jogging by the</td>
</tr>
<tr>
<td></td>
<td>specified percentage value. A large percentage value makes the function</td>
</tr>
<tr>
<td></td>
<td>less sensitive.</td>
</tr>
<tr>
<td>Collision Detection Memory</td>
<td>Defines how much the robot moves in reversed direction on the path after</td>
</tr>
<tr>
<td></td>
<td>a collision, specified in seconds. If the robot moved fast before the</td>
</tr>
<tr>
<td></td>
<td>collision it will move away a larger distance than if the speed was slow.</td>
</tr>
<tr>
<td></td>
<td>Collision Detection Memory is by default set to 75 ms.</td>
</tr>
<tr>
<td>Manipulator Supervision</td>
<td>Turns the supervision for the loose arm detection on or off for IRB 340 and</td>
</tr>
<tr>
<td></td>
<td>IRB 360. A loose arm will stop the robot and cause an error message.</td>
</tr>
<tr>
<td>Manipulator Supervision Level</td>
<td>Modifies the supervision level for the loose arm detection for the</td>
</tr>
<tr>
<td></td>
<td>manipulators IRB 340 and IRB 360. A large value makes the function less</td>
</tr>
<tr>
<td></td>
<td>sensitive.</td>
</tr>
</tbody>
</table>

Motion Planner
These parameters belong to the type Motion Planner in the topic Motion.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion Supervision Max Level</td>
<td>Set the maximum level to which the total collision detection tune level can</td>
</tr>
<tr>
<td></td>
<td>be changed. It is by default set to 300%.</td>
</tr>
</tbody>
</table>
### Motion System

This parameter belongs to the type *Motion System* in the topic *Motion*.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ind collision stop without</td>
<td>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.</td>
</tr>
<tr>
<td>brake</td>
<td></td>
</tr>
</tbody>
</table>

### General RAPID

These parameters belong to the type *General RAPID* in the topic *Controller*.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collision Error</td>
<td>Enables RAPID error handling for collision. <em>Collision Error Handler</em> is default set to Off. For more information regarding error handling for a collision, see <em>Technical reference manual - RAPID kernel</em></td>
</tr>
<tr>
<td>Handler</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
</table>
| MotionSup   | MotionSup is used to:  
  • activate or deactivate Collision Detection. This can only be done if the parameter Path Collision Detection is set to On.  
  • modify the supervision level with a specified percentage value (1-300%). A large percentage value makes the function less sensitive. |
4.2.5.3 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.

<table>
<thead>
<tr>
<th>Digital output</th>
<th>Description</th>
</tr>
</thead>
</table>
| MotSupOn       | MotSupOn is high when Collision Detection is active and low when it is not active. Note that a change in the state takes effect when a motion starts. Thus, if Collision Detection is active and the robot is moving, MotSupOn is high. If the robot is stopped and Collision Detection is turned off, MotSupOn is still high. When the robot starts to move, MotSupOn switches to low. Before the first Motors On order after a restart of the robot controller, MotSupOn will reflect the value of the corresponding system parameter Path Collision Detection:  
  • If Path Collision Detection is set to On, MotSupOn will be high.  
  • If Path Collision Detection is set to Off, MotSupOn will be low. |
| MotSupTrigg    | MotSupTrigg goes high when the collision detection triggers. It stays high until the error code is acknowledged from the FlexPendant. |
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 217*.

**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;
```
4.2.6.4 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

<table>
<thead>
<tr>
<th>If ...</th>
<th>then ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the payload is incorrectly defined</td>
<td>use Load Identification to define it. For more information, see Operating manual - OmniCore.</td>
</tr>
<tr>
<td>the payload has large mass or inertia</td>
<td>increase supervision level</td>
</tr>
<tr>
<td>the arm load (cables or similar) cause trigger</td>
<td>manually define the arm load or increase supervision level</td>
</tr>
<tr>
<td>the application involves many external process forces</td>
<td>increase the supervision level for jogging and program execution in steps of 30 percent until you no longer receive the error code.</td>
</tr>
<tr>
<td>the external process forces are only temporary</td>
<td>use the instruction MotionSup to raise the supervision level or turn the function off temporarily.</td>
</tr>
<tr>
<td>everything else fails</td>
<td>turn off Collision Detection.</td>
</tr>
</tbody>
</table>
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).

Besides the robot itself the function will monitor up 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 into 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.

Limitations

Paint robots, IRB 6620LX, and IRB 360 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. The Collision Avoidance functionality between 2 robots (or more) can only be achieved when using a MultiMove system.

⚠️ CAUTION

Collision Avoidance shall not be used for safety of personnel.

Continues on next page
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 226.
- Decrease the general safety distance with the system parameter Coll-Pred Safety Distance.

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.
4.4 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.

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.

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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SafeMove Assistance Speed Factor</td>
<td>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.</td>
</tr>
<tr>
<td>SafeMove assistance zone margin</td>
<td>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.</td>
</tr>
</tbody>
</table>
For more information, see the parameters in the type \textit{Motion System} described in \textit{Technical reference manual - System parameters}.
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 233*.

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 loose 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
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:

- SafeMove
- Track Motion (IRBT)
- Positioners (IRBP) on Interchange axes

`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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent Joint</td>
<td>Flag that determines if independent mode is allowed for the axis.</td>
</tr>
<tr>
<td>Independent Upper Joint Bound</td>
<td>Defines the upper limit of the working area for the joint when operating in independent mode.</td>
</tr>
<tr>
<td>Independent Lower Joint Bound</td>
<td>Defines the lower limit of the working area for the joint when operating in independent mode.</td>
</tr>
</tbody>
</table>

Transmission

These parameters belong to the type Transmission in the topic Motion.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission Gear High</td>
<td>Independent Axes requires high resolution in transmission gear ratio, which is therefore defined as Transmission Gear High divided by Transmission Gear Low. If no smaller number can be used, the transmission gear ratio will be correct if Transmission Gear High is set to the number of cogs on the robot axis side, and Transmission Gear Low is set to the number of cogs on the motor side.</td>
</tr>
<tr>
<td>Transmission Gear Low</td>
<td>See Transmission Gear High.</td>
</tr>
</tbody>
</table>
5 Motor Control

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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndAMove</td>
<td>IndAMove (Independent Absolute position Movement) change an axis to independent mode and move the axis to a specified position.</td>
</tr>
<tr>
<td>IndCMove</td>
<td>IndCMove (Independent Continuous Movement) change an axis to independent mode and start moving the axis continuously at a specified speed.</td>
</tr>
<tr>
<td>IndDMove</td>
<td>IndDMove (Independent Delta position Movement) change an axis to independent mode and move the axis a specified distance.</td>
</tr>
<tr>
<td>IndRMove</td>
<td>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.</td>
</tr>
<tr>
<td>IndReset</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndInpos</td>
<td>IndInpos indicates whether an axis has reached the selected position.</td>
</tr>
<tr>
<td>IndSpeed</td>
<td>IndSpeed indicates whether an axis has reached the selected speed.</td>
</tr>
</tbody>
</table>
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
!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 (±180°).

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

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*.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pathrecid</td>
<td>pathrecid is used to identify a breakpoint for the path recorder.</td>
</tr>
</tbody>
</table>

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*.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StorePath</td>
<td>StorePath is used to store the movement path being executed when an error or interrupt occurs. StorePath is included in RobotWare base.</td>
</tr>
<tr>
<td>RestoPath</td>
<td>RestoPath is used to restore the path that was stored by StorePath. RestoPath is included in RobotWare base.</td>
</tr>
<tr>
<td>PathRecStart</td>
<td>PathRecStart is used to start recording the robot’s path. The path recorder will store path information during execution of the robot program.</td>
</tr>
<tr>
<td>PathRecStop</td>
<td>PathRecStop is used to stop recording the robot’s path.</td>
</tr>
<tr>
<td>PathRecMoveBwd</td>
<td>PathRecMoveBwd is used to move the robot backwards along a recorded path.</td>
</tr>
<tr>
<td>PathRecMoveFwd</td>
<td>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.</td>
</tr>
<tr>
<td>SyncMoveSuspend</td>
<td>SyncMoveSuspend is used to suspend synchronized movements mode and set the system to independent movement mode.</td>
</tr>
<tr>
<td>SyncMoveResume</td>
<td>SyncMoveResume is used to go back to synchronized movements from independent movement mode.</td>
</tr>
</tbody>
</table>

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*.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PathRecValidBwd</td>
<td>PathRecValidBwd is used to check if the path recorder is active and if a recorded backward path is available.</td>
</tr>
<tr>
<td>PathRecValidFwd</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 robtarget 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;
```
... 
!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.

! 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.
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.

```rapid
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 chooses 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
```
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.
For more information, see the section about `PathRecStop` in *Technical reference manual - RAPID Instructions, Functions and Data types*.

```plaintext
...  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.

![Diagram of multitasking](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
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 247.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>The name of the task. 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. 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.</td>
</tr>
<tr>
<td>Task in foreground</td>
<td>Used to set priorities between tasks. <em>Task in foreground</em> 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. If <em>Task in foreground</em> is set to empty string for a task, it runs at the highest level.</td>
</tr>
</tbody>
</table>
| Type               | Controls the start/stop and system restart behavior:  
|                    | - 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 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.  
|                    | A task that controls a mechanical unit must be of the type *normal*. |
| 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         | *TrustLevel* defines the system behavior when a static or semistatic task program is stopped (e.g. due to error):  
|                    | - 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. |
6 RAPID Program Features

6.2.2 System parameters

Continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MotionTask</td>
<td>Indicates whether the task program can control robot movement with RAPID move instructions. Only one task can have MotionTask set to YES unless the option MultiMove is used.</td>
</tr>
</tbody>
</table>
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*.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskid</td>
<td>Identify available tasks in the system. This identity is defined by the system parameter Task, and cannot be defined in the RAPID program. However, the data type taskid can be used as a parameter when declaring a routine. For code example, see <a href="#">taskid on page 258</a>.</td>
</tr>
<tr>
<td>syncident</td>
<td>Used to identify the waiting point in the program, when using the instruction WaitSyncTask. The name of the syncident variable must be the same in all task programs. For code example, see <a href="#">WaitSyncTask example on page 252</a>.</td>
</tr>
<tr>
<td>tasks</td>
<td>A variable of the data type tasks contains names of the tasks that will be synchronized by the instruction WaitSyncTask. For code example, see <a href="#">WaitSyncTask example on page 252</a>.</td>
</tr>
</tbody>
</table>

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*.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitSyncTask</td>
<td>WaitSyncTask is used to synchronize several task programs at a special point in the program. A WaitSyncTask 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 <a href="#">WaitSyncTask example on page 252</a>.</td>
</tr>
</tbody>
</table>

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*.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestAndSet</td>
<td>TestAndSet 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 <a href="#">Example with flag and TestAndSet on page 256</a>.</td>
</tr>
<tr>
<td>TaskRunMec</td>
<td>Check if the task program controls any mechanical unit (robot or other unit). For code example, see <a href="#">Test if task controls mechanical unit on page 257</a>.</td>
</tr>
<tr>
<td>TaskRunRob</td>
<td>Check if the task program controls any robot with TCP. For code example, see <a href="#">Test if task controls mechanical unit on page 257</a>.</td>
</tr>
</tbody>
</table>
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:**

```rapid
MODULE module1
  PERS bool startsync:=FALSE;
PERS string stringtosend:="";
PROC main()
  stringtosend:="this is a test";
  startsync:= TRUE
ENDPROC
ENDMODULE
```

**Background task program:**

```rapid
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
```
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.
6.2.4.2 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 \texttt{WaitUntil} or \texttt{WHILE}.

If the instruction \texttt{WaitUntil} is used, it will poll internally every 100 ms.

\begin{center}
\textbf{CAUTION}
\end{center}

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

\begin{verbatim}
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
\end{verbatim}

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:**

```rapid
MODULE module1
PROC main()
  SetDO do1,1;
  ...
ENDPROC
ENDMODULE
```

**Background task program:**

```rapid
MODULE module2
VAR intnum intno1;

PROC main()
  CONNECT intno1 WITH wait_trap;
  ISignalDO do1, 1, intno1;
  WHILE TRUE DO
    WaitTime 10;
  ENDWHILE
ENDPROC

TRAP wait_trap
  ! This is the point where the execution
  ! continues after do1 is set in main task
  ...
  IDelete intno1;
ENDTRAP
ENDMODULE
```
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 robot's 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:

```rapid
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:

```rapid
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
6.2.4.4 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:

```rapid
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:

```rapid
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
6 RAPID Program Features

6.2.4.4 Using a dispatcher

Continued

ENDPROC
ENDMODULE
6 RAPID Program Features

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:

```plaintext
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:

```plaintext
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 .

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 245.

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
endif
6.2.5.3 taskid

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

```rapid
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.
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

External computer

Ethernet TCP/IP

Description

Several robots can access the same computer over an Ethernet network.

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.

SFTP supports the following servers:

- Rebex version 1.0.3

Continues on next page
7 Communication

7.1.1 Introduction to FTP&SFTP client

Continued

- 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

- When using the FTP client the maximum length for a file name is 99 characters.
- When using the FTP client the maximum length for a file path including the file name is 200 characters. The whole path is included in the 200 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.
- When using the SFTP 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.

Example FTP

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>myFTP</td>
</tr>
<tr>
<td>Server path</td>
<td>C:\robot_1</td>
</tr>
</tbody>
</table>

- A backup is saved to myFTP/Backups/Backup_20130109 (27 characters)
- The path on the PC will be C:\robot_1\Backups\Backup_20130109 (34 characters)

Continues on next page
The longest file path inside this backup is 
\texttt{C:/robot_1/Backups/Backup_{20130109}/RAPID/TASK1/PROGMOD/myprogram.mod}  
\texttt{(54+13 characters)}

The maximum path length for this example first looks like 27 characters but is actually 67 characters.

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Parameter} & \textbf{Value} \\
\hline
Name & mySFTP \\
\hline
\end{tabular}
\end{center}

- A backup is saved to \texttt{mySFTP/Backups/Backup_{20130109}}  
\texttt{(27 characters)}

- The path on the PC will be \texttt{/Backups/Backup_{20130109}}  
\texttt{(24 characters)}

- The longest file path inside this backup is \texttt{/Backups/Backup_{20130109}/RAPID/TASK1/PROGMOD/myprogram.mod}  
\texttt{(44+13 characters)}

The maximum path length for this example first looks like 27 characters but is actually 57 characters.

\section*{System parameters}

See \textit{Technical reference manual - System parameters}.
7 Communication

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. 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.

Example

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>myNFS</td>
</tr>
<tr>
<td>Server path</td>
<td>C:\robot_1</td>
</tr>
</tbody>
</table>

Continues on next page
• A backup is saved to myNFS/Backups/Backup_20130109
  (27 characters)
• The path on the PC will be C:\robot_1\Backups\Backup_20130109
  (34 characters)
• The longest file path inside this backup is
  C:\robot_1\Backups\Backup_20130109\RAPID\TASK1\PROGMOD\myprogram.mod
  (54+13 characters)

The maximum path length for this example first looks like 27 characters but is
actually 67 characters.
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

**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 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 276.

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

## 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*.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>corrdescr</td>
<td><em>corrdescr</em> is a correction generator descriptor that is used as the reference to the correction generator.</td>
</tr>
</tbody>
</table>

## 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*.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CorrCon</td>
<td><em>CorrCon</em> activates path correction. Calling <em>CorrCon</em> 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).</td>
</tr>
<tr>
<td>CorrDiscon</td>
<td><em>CorrDiscon</em> deactivates path correction. Calling <em>CorrDiscon</em> will disconnect a correction generator.</td>
</tr>
<tr>
<td>CorrClear</td>
<td><em>CorrClear</em> deactivate path correction. Calling <em>CorrClear</em> will disconnect all correction generators.</td>
</tr>
<tr>
<td>CorrWrite</td>
<td><em>CorrWrite</em> sets the path correction values. Calling <em>CorrWrite</em> will set the offset values to a correction generator.</td>
</tr>
</tbody>
</table>

## 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*.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CorrRead</td>
<td><em>CorrRead</em> reads the total correction made by a correction generator.</td>
</tr>
</tbody>
</table>
9.2.3 Related RAPID functionality

The 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

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

```plaintext
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 I/O configuration file.

Activate Auto Acknowledge Input

The robot system must be installed with the option Auto Acknowledge Input enabled in Installation Manager.

Use the following procedure to activate the system input for Auto Acknowledge Input.

<table>
<thead>
<tr>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
| 2      | Edit the I/O configuration file, eio.cfg, using a text editor. Add the following line in the group SYSSIG_IN:  
- Signal "my_signal_name" -Action "AckAutoMode"  
my_signal_name 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. |
Index

3
3rd party software, 13

A
Absolute Accuracy, 159
Absolute Accuracy calibration, 170
Absolute Accuracy compensation, 168
Absolute Accuracy verification, 171
acknowledge messages, 153
activate Absolute Accuracy, 162
activate supervision, 223
actor signals, 135–136
Advanced RAPID, 17
Advanced Shape Tuning, 180
AliasIO, 24–25
alignment, 174
analog signal, 48
Analog Signal Interrupt, 48
AND, 136
ArgName, 46
argument name, 46
arm replacement, 164
Auto acknowledge input, 12, 277
automatic friction tuning, 181
axis, 229
axis reset, 229

B
binary communication, 117
binary data, 153
birth certificate, Absolute Accuracy, 172
BitAnd, 19
BitCheck, 19
BitClear, 19
bit functionality, 18
BitLSh, 19
BitNeg, 19
BitOr, 19
BitRSh, 19
BitShift, 19
BitXOr, 19
BookErrNo, 41
byte, 19
ByteToStr, 19

C
calibrate tool, 178
calibration data, 162
calibration process, 170
calibration tools, 161
CalibWare, 161
cell alignment, 174
certificate, Absolute Accuracy, 172
change calibration data, 162
correlation based communication, 117
Check unresolved references, Task type, 245
CirPathMode, 200
ClearRawBytes, 122
Close, 118
CloseDir, 126
collision, 214
Collision Avoidance, 225
collision detection
YuMi robots, 212
Collision Detection Memory, 217
Collision Error Handler, 218
common data, 249
communication, 116
compensation, 168
compensation parameters, 159, 173
compliance errors, 167
configuration
Absolute Accuracy, 162
configuration functionality, 27
configure Collision Detection, 221
coordinate systems, 174
CopyFile, 126
CopyRawBytes, 122
Corr argument, 275
CorrClear, 274
CorrCon, 274
corrdescr, 274
CorrDiscon, 274
correction generator, 272
CorrRead, 274
CorrWrite, 274
cross connections, 135
cut plane, 198
cut shape, 203
Cyclic bool, 104
Cyclic bool settings, 110
Cyclic bool system parameters, 110

data, 141
datapos, 22
data search functionality, 21
data types
Multitasking, 247
deactivate supervision, 223
declarations, 249
deflection, 168
digital I/O signals, 135
dir, 126
directory management, 125
discarded message, 143
dispatcher, 254

errdomain, 38
error interrupts, 37
error sources in accuracy, 167
ErrRaise, 38
errtype, 38
Ethernet, 261, 264
event messages, 40
event number, 40
Event Preset Time, 132
external axes, 213
external axis, 229

F
fake target, 168
false triggering, 224
Fieldbus Command Interface, 112
FIFO, 142
file communication, 116
file management, 125
FileSize, 126
file structures, 125
fixed position events, 129
Index

fixture alignment, 175
FlexPendant, 256
frame relationships, 177
frames, 174
FricdEvaluate, 187
FricdInit, 187
FricdSetFricLevels, 187
friction compensation, 180
Friction FFW Level, 185
Friction FFW On, 185
Friction FFW Ramp, 185
friction level tuning, 181
FSSize, 126

functions
  Advanced RAPID, 46
  Multitasking, 247

G
General RAPID, 218
GetDataVal, 22
GetMaxNumberOfCyclicBool, 111
GetNextCyclicBool, 111
GetNextSym, 22
GetNumberOfCyclicBool, 111
GetTrapData, 38

I
iError, 38
IndAMove, 232
IndCMove, 232
Ind collision stop without brake, 218
IndDMove, 232
Independent Axes, 229
independent joint, 213
Independent Joint, 231
Independent Lower Joint Bound, 231
Independent movement, 229
Independent Upper Joint Bound, 231
IndInpos, 232
IndReset, 232
IndRMove, 232
IndSpeed, 232

instructions
  Advanced RAPID, 46
  Multitasking, 247
interrupt, 48, 142, 250
interrupt functionality, 37
iodev, 118
IPers, 38
IRMQMessage, 146
IsCyclicBool, 111
IsFile, 126
ISignalAI, 49
ISignalAO, 49
IsStopStateEvent, 46

J
Jog Collision Detection, 217, 221
Jog Collision Detection Level, 217
Jog Collision Detection Level, 221
joint zones, 205

K
kinematic errors, 167

L
licenses, 13
load calibration data, 162
Load Identification, 161
logical AND, 137
Logical Cross Connections, 135
logical operations, 135
logical OR, 137
loss of accuracy, 166
lost message, 143
lost queue, 143

M
Main entry, Task type, 245
maintenance, 164
MakeDir, 126
manipulator replacement, 165
Manipulator Supervision, 217
Manipulator Supervision Level, 217
manual friction tuning, 183
measurement system, 232
mechanical unit, 257
merge of messages, 153
Motion Planner, 217
Motion Process Mode, 188
MotionSup, 219, 223
Motion Supervision, 217
Motion Supervision Max Level, 217
Motion System, 218
MotionTask, Task type, 246
motor replacement, 164
MotSupOn, 220
MotSupTrigg, 220
MoveCSync, 131
MoveJSync, 131
MoveLSync, 131
Multitasking, 243

N
NFS Client, 264
non printable characters, 153
NORMAL, 245
NoSafety, 245
NOT, 137

O
Open, 118
OpenDir, 126
open source software, OSS, 13
OR, 136

P
PackDNHeader, 113
PackRawBytes, 122
parameters
  accuracy compensation, 173
  path, 31
Path Collision Detection, 217, 221
Path Collision Detection Level, 217, 221
path correction, 272
path offset, 272
pathrecid, 236
PathRecMoveBwd, 236
PathRecMoveFwd, 236
path recorder, 239
Path Recovery, 235
PathRecStart, 236
queue handling, 142
queue name, 142

R
r1_calib, 162
RAPID components
Advanced RAPID, 46
Multitasking, 247
RAPID Message Queue, 140
RAPID support functionality, 45
rawbytes, 122
RawBytesLen, 122
raw data, 121
ReadAnyBin, 118
ReadBin, 118
ReadCfgData, 28
ReadDir, 126
ReadErrData, 38
ReadNum, 118
ReadRawBytes, 122
ReadStr, 118
ReadStrBin, 118
record, 141
recorded path, 239
recover path, 235
RemoveAllCyclicBool, 111
RemoveCyclicBool, 111
RemoveDir, 126
RemoveFile, 126
RenameFile, 126
replacements, 164
reset, 232
reset axis, 229
resolver offset calibration, 170
restartdata, 34
RestoPath, 236
resultant signal, 135–136
resume signals, 35
reversed movement, 214
Rewind, 118
RMQEmptyQueue, 146
RMQFindSlot, 146
RMQGetMessage, 146
RMQGetMsgData, 146
RMQGetMsgHeader, 146
RMQGetSlotName, 146
rmqheader, 146
RMQ Max Message Size, 145
RMQ Max No Of Messages, 145
rmqmessage, 146
RMQ Mode, 145
RMQReadWait, 146
RMQSendMessage, 146
RMQSendWait, 146
rmqslot, 146
RMQ Type, 145
robot alignment, 176
roll, 167
routine call, 254

S
SafeMove Assistant, 227
SEMISTATIC, 245
sensor, 272
SetAllDataVal, 22
SetDataSearch, 22
SetDataVal, 22
SetSysData, 46
set up Collision Detection, 221
SetUpCyclicCollision, 111
shapedata, 207
shared resources, 256
signal, 250, 254
SocketAccept, 154
SocketBind, 154
SocketClose, 154
SocketConnect, 154
SocketCreate, 154
socketdev, 154
SocketGetStatus, 155
SocketListen, 154
Socket Messaging, 151
SocketReceive, 154
SocketSend, 154
socketstatus, 154
soft servo, 213
software licenses, 13
speed, 215
STATIC, 245
stationary world zone, 207
StepBwdPath, 34
StoreBwdPath, 236
string termination, 153
StrToByte, 19
supervision level, 217, 219, 223
synchronizing tasks, 252
syncident, 252
syncident, data type, 247
SyncMoveResume, 236
SyncMoveSuspend, 236
SysFail, 245
SysHalt, 245
SysStop, 245
system parameters
configuration functionality, 27
Multitasking, 245
system resources, 256

T
Task, Task type, 245
Task, type, 245
taskid, 247, 258
taskid, data type, 247
Task in foreground, Task type, 245
TaskRunMec, 257
TaskRunMec, function, 247
TaskRunRob, 257
TaskRunRob, function, 247
tasks, 243, 252
data type, 247

tasks, data type, 247
temporary world zone, 207
TestAndSet, 256
TestAndSet, function, 247
TextGet, 41
TextTabFreeToUse, 41
TextTabGet, 41
TextTabInstall, 41
text table file, 40
tool calibration, 178
tools, 161
torque, 215

track motion, 213
Transmission Gear High, 231
Transmission Gear Low, 231
trapdata, 38

trap routine, 142
TriggC, 131
TriggCheckIO, 131
triggdata, 130
triggering, 224
Trigglnt, 130
TriggIO, 130
triggios, 130
triggiosdnum, 130
TriggJ, 131
TriggL, 131
TriggLIOs, 131
TriggRampAO, 131
TriggSpeed, 34
TriggStopProc, 34
triggstrgo, 130

TrustLevel, Task type, 245
TUNE_FRIC_LEV, 183
TUNE_FRIC_RAMP, 183
TuneServo, 183
tuning, 223
tuning, automatic, 181
tuning, manual, 183
Type, Task type, 245

U
uncalib, 162
UnpackRawBytes, 122
user message functionality, 40
Use Robot Calibration, 162

V
verification, 171

W
waiting for tasks, 252
WaitSyncTask, 252
WaitSyncTask, instruction, 247
WaitUntil, 250
WarmStart, 28
world zones, 205
Wrist Move, 196
wrist replacement, 164
Write, 118
WriteAnyBin, 118
WriteBin, 118
WriteCfgData, 28
WriteRawBytes, 122
WriteStrBin, 118
WZBoxDef, 207
WZCylDef, 207
WZDisable, 208
WZDOSet, 208
WZDOSet, 208
WZEnable, 208
WZFree, 208
WZHomeJointDef, 208
WZLimJointDef, 208
WZLimSup, 208
WZSpDef, 207
wzstationary, 207
wzttemporary, 207

Y
yaw, 167

Z
zones, 205