ROBOTICS

Application manual

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

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

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

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

Keep for future reference.

Additional copies of this manual may be obtained from ABB.

Original instructions.

© Copyright 2020-2022 ABB. All rights reserved.
Specifications subject to change without notice.
Table of contents

Overview of this manual ................................................................. 7
Safety .............................................................................................. 9

1 Getting started ............................................................................. 11
  1.1 About RobotWare add-ins .......................................................... 11
  1.2 Add-in directory and file structure ............................................... 12
  1.3 Quick start procedures for example add-in ............................. 14

2 Reference material ....................................................................... 19
  2.1 Custom event log messages ......................................................... 19
    2.1.1 About event log messages ..................................................... 19
    2.1.2 Event log texts ..................................................................... 20
    2.1.3 Validating event log .xml files ............................................. 22
    2.1.4 Configure event logs to take focus on the FlexPendant ......... 23
  2.2 System parameters related to add-in development ................. 25
    2.2.1 About cfg files ..................................................................... 25
    2.2.2 Topic Controller .................................................................. 28
    2.2.3 Topic I/O System .................................................................. 31
    2.2.4 Topic Man-machine Communication .................................... 32
    2.2.5 Example cfg files .................................................................. 42
  2.3 The install.cmd file .................................................................... 44
    2.3.1 Introduction .......................................................................... 44
    2.3.2 Commands ............................................................................ 48
    2.3.3 Examples of install.cmd files ............................................... 61
  2.4 RAPID ..................................................................................... 62
    2.4.1 RAPID modules .................................................................... 62
    2.4.2 Using text resources from files .......................................... 64
    2.4.3 Hiding RAPID content ........................................................ 65
    2.4.4 Optional settings for RAPID arguments (RAPID meta data) ... 67
      2.4.4.1 Hiding arguments in programs ....................................... 68
      2.4.4.2 Hiding optional argument when changing selected instruction 70
      2.4.4.3 Argument filter ............................................................... 72
      2.4.4.4 Argument value range .................................................... 74
  2.5 RobotWare Add-In Packaging tool ............................................. 75
    2.5.1 Introduction .......................................................................... 75
      2.5.1.1 About the RobotWare Add-In Packaging tool .................. 75
      2.5.1.2 Optional features ............................................................ 77
      2.5.1.3 Files of a packaged add-in ............................................. 79
      2.5.1.4 Signing with digital certificates ..................................... 80
      2.5.1.5 Types of add-in packaging tools .................................... 84
    2.5.2 User interface ....................................................................... 85
      2.5.2.1 The home page ............................................................... 85
      2.5.2.2 The File menu ............................................................... 86
      2.5.2.3 The Product Manifest view .......................................... 88
    2.5.3 Creating and building an add-in project ............................ 100
    2.5.4 Building an add-in from the console ................................. 101

2.6 License Generator ..................................................................... 103
  2.6.1 Introduction .......................................................................... 103
  2.6.2 The user interface ................................................................. 104
    2.6.2.1 The Preferences window ............................................... 104
    2.6.2.2 The main window .......................................................... 105
  2.6.3 Creating the license ............................................................... 107

A Appendix: Migration from RobotWare 6 .................................. 109

Index ......................................................................................... 113
This page is intentionally left blank
Overview of this manual

About this manual
This manual contains instructions for how to create your own add-in to use with ABB robots. The manual is divided into two parts:

- **Getting started:**
  This section provides examples and instructions to enable you to create and start using a simple, but completely functioning, add-in with little effort.

- **Reference material:**
  This section provides the information you need in order to be able to further develop your add-in. The reference material includes information such as system parameters, commands and RAPID meta data, as well as instructions for the tools used for packaging and licensing.

**Note**
Chapters under section *Reference material on page 19* are to the large extent independent of each other and can be read in any order.

It is recommended to read section *Getting started on page 11* and try the example before reading specific chapters in the reference part.

The reference part contains information relevant to RobotWare 7, without going into details of differences between RobotWare 6 and RobotWare 7. The appendix *Appendix: Migration from RobotWare 6 on page 109* contains information that can be useful for add-in migration from RobotWare 6 to RobotWare 7.

Usage
With the help of this manual, you can learn how to create functionality that extends the base RobotWare system, so called RobotWare add-ins. You will also learn how to package and distribute these add-ins.

Who should read this manual?
This manual is intended for:

- Line builders that want to implement the same program solution on many robots
- Value providers, selling the ABB robots with their own functionality added
- ABB companies selling robots

Prerequisites
The reader should...

- be experienced in working with ABB robots
- be experienced RAPID programmer
- be familiar with system parameters
- have the latest version of the RobotWare Add-In Packaging tool that supports RobotWare 7 (minimal tool version is 1.10)

*Continues on next page*
Overview of this manual

Continued

References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Document ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical reference manual - System parameters</td>
<td>3HAC065041-001</td>
</tr>
<tr>
<td>Operating manual - RobotStudio</td>
<td>3HAC032104-001</td>
</tr>
<tr>
<td>Operating manual - Integrator's guide OmniCore</td>
<td>3HAC065037-001</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 7.1.</td>
</tr>
<tr>
<td>B</td>
<td>Released with RobotWare 7.2.</td>
</tr>
<tr>
<td></td>
<td>• All information about <code>eventlogtitle</code> removed from sections Add-in directory and file structure on page 12, Custom event log messages on page 19 and Commands on page 48.</td>
</tr>
<tr>
<td></td>
<td>• New script <code>math_lib_set_mem_size</code> added in section Commands on page 48.</td>
</tr>
<tr>
<td>C</td>
<td>Released with RobotWare 7.5.</td>
</tr>
<tr>
<td></td>
<td>• Section Argument Name Rules (MMC_REAL_PARAM) on page 34 updated with information about how to add a string in Rapid rules.</td>
</tr>
<tr>
<td>D</td>
<td>Released with RobotWare 7.7.</td>
</tr>
<tr>
<td></td>
<td>• Information regarding the <code>config</code> command updated in section Commands on page 48.</td>
</tr>
<tr>
<td></td>
<td>• Updated the section Building an add-in from the console on page 101.</td>
</tr>
</tbody>
</table>
Safety

Safety of personnel

A robot is heavy and extremely powerful regardless of its speed. A pause or long stop in movement can be followed by a fast hazardous movement. Even if a pattern of movement is predicted, a change in operation can be triggered by an external signal resulting in an unexpected movement.

Therefore, it is important that all safety regulations are followed when entering safeguarded space.

Safety regulations

Before beginning work with the robot, make sure you are familiar with the safety regulations described in the manual *Safety manual for robot - Manipulator and IRC5 or OmniCore controller.*
1 Getting started

1.1 About RobotWare add-ins

What is an add-in

Add-ins are independently developed and versioned software packages that extend the capabilities offered by RobotWare, making ABB's robot controllers even smarter and even more user-friendly. Creating RobotWare add-ins is also the recommended way for 3rd party developers to add new features into RobotWare.

An add-in can include several RAPID modules, system modules, or program modules which hold the basic code for the add-in. The add-in also includes script files for initializing the add-in functionality at start-up. The add-in may also include .xml files with custom-defined event log messages in different languages.

An add-in may also implement one or more FlexPendant applications using the WebApps concept (introduced in RobotWare 7). This manual covers the controller side implementation and the packaging of the add-in contents for distribution. For more information on how to implement FlexPendant applications, see the OmniCore App SDK manual, available as part of the SDK download.

Packaging your add-in

Once the content of an add-in is developed, it needs to be packaged so that it can be distributed and installed into a RobotWare system. RobotWare add-ins use the ABB proprietary format that is called rpk-format. The RobotWare Add-In Packaging tool is used to produce such a package. The tool also produces the package metadata file, the manifest file, which is also a part of the package.

Licensing your add-in

The add-in can have one of the following license models:

- Make the add-in available to anyone without charge (open add-in).
- Require a license for using the add-in.

To package and distribute a simple unlicensed/open add-in, the only tool needed is the RobotWare Add-In Packaging tool. That is the simplest way to get started.

When working with licensed add-ins, you also need the tool License generator. The License generator is not needed for creating and packaging a licensed add-in, but it is needed for to create the licenses that open the add-in functionality to the users that will install and use the developed add-in.
Recommended file structure

An add-in implementation consists of several files and directories. The following structure is recommended:

```
<option name>

install.cmd
...

config

language

install.cmd

instlang.cmd

en

elogtext.xml
text.xml
text_utf8.xml
...
...
...

RAPID

<...>.sys
<...>.mod

WebApps
```

Add-in files

In order to make your own add-in, the following files must be created:

<table>
<thead>
<tr>
<th>File type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>install.cmd</td>
<td>Installation script. Specifies for example which .cfg files to load, see The install.cmd file on page 44. Must be in the root directory of your add-in.</td>
</tr>
<tr>
<td>File type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>config/</td>
<td>One or several <code>.cfg</code> files with the configuration of system parameters. If the add-in includes RAPID, one of the <code>.cfg</code> files should specify which RAPID module (.sys file) to load, see System parameters related to add-in development on page 25.</td>
</tr>
<tr>
<td>language/</td>
<td>Contains installation scripts and language specific resource files. For more information about implementing custom event log messages, see Custom event log messages on page 19. It is possible to create and use language specific text resource files that can be used from RAPID programs. For more information, see Including language files from your add-in on page 64.</td>
</tr>
<tr>
<td>RAPID/</td>
<td>The RAPID source code of your modules contained in files .sys or .mod, see RAPID modules on page 62.</td>
</tr>
<tr>
<td>WebApps</td>
<td>The WebApps directory can be empty or contain one or more subdirectories containing FlexPendant web applications. Each application must be stored in its own subdirectory.</td>
</tr>
</tbody>
</table>
1.3 Quick start procedures for example add-in

Prerequisites

- RobotWare Add-In Packaging tool. Download from ABB Library Download Center.
- RobotStudio

Copy the example add-in

1. Locate the example subdirectory of the installation directory of RobotWare Add-In Packaging tool (for example C:/Program Files (x86)/ABB Industrial IT/Robotics IT/AddinPackagingTool/Examples).
2. Copy the Circlemove7 folder from this location and place it on your local disk (for example C:/Users/MyName/AddIns/Circlemove7).

Get familiar with the contents of the add-in

Once when you have copied the add-in, inspect the Circlemove7 contents.

Note that the Circlemove7 example has three additional files in the root directory of the example (Circlemove.manifest, Circlemove.rpkproj and Circlemove.rpkspec). These files are not a part of the add-in implementation, but are used by the add-in packaging tool to store information about the add-in (such as name, version etc.) which is necessary for packaging the add-in.

If you wish to change/customize the example project, then edit the example files. See Reference material on page 19 for detailed information on how to work with specific topics. If not, you can just proceed reading this chapter and come back to the specific topic later.

The add-in includes the following functionality:

- Implements a RAPID command called Circlemove, which can be used by RAPID programs.
- Shows how to define and use custom event log messages.
- Shows how to use localizable text resources from your RAPID code.
- Implements a simple WebApp that shows up on the FlexPendant. No additional configuration is needed. For the application to be loaded properly on the FlexPendant, the add-in registers itself using the register-type option command in the install.cmd file, see register on page 57.

See Add-in directory and file structure on page 12 for location of the files.
Package your add-in for installation

To make it possible to install and use the add-in in a RobotWare system, it must first be packaged. This is done using the RobotWare Add-In Packaging tool. To open the add-in in the packaging tool, simply double-click on the *Circlemove.rpkproj* file which is located in the root directory of the example. When the project is open, click on **Build>Build project**.

![Image of the packaging tool with selected Build Project option](image)

The result of the **Build** process is a directory containing two files:

- *Rmf file* – the manifest (metadata) of the add-in package
- *Rpk file* – the implementation of the add-in

Create a Virtual Controller in RobotStudio using Installation Manager 7

**Installation Manager 7**

The add-in is included in the Virtual Controller system by browsing to the *Circlemove.rmf* file in the **Product** step of the system creation. See *Operating manual - Integrator’s guide OmniCore* for more information about adding products and using Installation Manager.
Try the add-in

To verify that your add-in is properly installed in the virtual system created in the previous step, do the following in RobotStudio:

1 Select your Virtual Controller and start the Omnicore virtual FlexPendant that is connected to your virtual system. The Circlemove user interface shall appear on the start page of the FlexPendant:
2 Click on the Circlemove icon and the following page showing a rotating ABB logo shall appear:

![Circlemove logo](image1.jpg)

3 Go back to the main page and open the RAPID editor (create a program if necessary). Verify that the MoveCircle RAPID instruction is available:

![RAPID editor with MoveCircle instruction](image2.jpg)
Distributing add-in to your users

Once you are satisfied with your add-in and you wish to distribute it to other users, this can be done in two different formats:

- **Rpk/rmf format**
  This is the output that the add-in packaging tool produces. As seen previously in this example, this is the input format required by the Installation Manager when creating a RobotWare system.

- **RobotStudio package, rspack format**
  In this case, the rpk/rmf files are packaged once again into an rspack file. Using this method, several add-ins can be included into a single rspack file. The rspack-s can be imported into RobotStudio using the Add-In page. All add-ins included in the rspack package will be automatically available to the Installation Manager when creating a RobotWare system. To learn more on how to work with RobotStudio rspack, see the RobotStudio developer center documentation:

2 Reference material

2.1 Custom event log messages

2.1.1 About event log messages

Overview

It is possible to create your own event log messages. The text of the message is placed in one .xml file for each language. You can then use RAPID instructions such as `ErrRaise` and `ErrLog` in the Circlemove example to raise an error using this message. Language independent strings can be used as arguments to `ErrRaise` and `ErrLog`, and be included in the message.

Event log message .xml file

The event log messages are added to the system via an .xml file that contains all the information about the messages. The file can be given any name, as long as the installation script `install.cmd` points out the correct file name. It is, however, recommended to use the following name:

- `<Add-In name>_elogtext.xml`

Template file

A template files for the required file `template_elogtext.xml` is included in the RobotWare installation. The template is located in the following directory in the RobotWare package folder, such as `...\ProgramData\ABB\DistributionPackages\ABB.RobotWare-<version>\RobotPackages\RobotControl_<version>\utility\Template\Elog`.

Note

Navigate to the RobotWare installation folder from the RobotStudio Add-Ins tab, by right-clicking on the installed RobotWare version in the Add-Ins browser and selecting Open Package Folder.
2.1.2 Event log texts

Overview

All event log messages must be written in the following .xml file:

- `<Add-In name>_elogtext.xml`

The messages must have unique numbers, within its domain, which are used to reference the message text from the RAPID code.

Explanation of the .xml file

This is a list of the XML tags and arguments that you need to define. All other tags and arguments should always look like in the example below. The complete syntax is also shown in the example below.

<table>
<thead>
<tr>
<th>XML tag or argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domainNo</td>
<td>Event log messages are divided into different domains. Domain number 8 is called User events and is reserved for non-ABB messages. For add-ins, always use domain 8 to avoid conflict with messages defined by ABB.</td>
</tr>
<tr>
<td>lang</td>
<td>Language code for the text in the messages. The same two-letter code as the name of the folder where the message .xml files are placed. This code is defined by the standard ISO 639.</td>
</tr>
<tr>
<td>min</td>
<td>The first message number in this file.</td>
</tr>
<tr>
<td>max</td>
<td>The last message number in this file.</td>
</tr>
<tr>
<td>Message number</td>
<td>Create one instance of Message for each error message.</td>
</tr>
<tr>
<td>number</td>
<td>A unique number, between 1 and 9999, identifying the error message. Make sure that the systems using this add-in will not have other add-ins using the same message numbers.</td>
</tr>
<tr>
<td>eDefine</td>
<td>A unique name for the message. Keep it short and descriptive.</td>
</tr>
<tr>
<td>Title</td>
<td>The message title that will be shown in the event log.</td>
</tr>
<tr>
<td>Description</td>
<td>The text describing the error, shown in the event log.</td>
</tr>
<tr>
<td>arg</td>
<td>A string used as argument in the ErrRaise or ErrLog instruction will be inserted in the message.</td>
</tr>
<tr>
<td>format</td>
<td>The format of the argument string from ErrRaise or ErrLog. For example %.40s means that the string cannot be longer than 40 characters.</td>
</tr>
<tr>
<td>ordinal</td>
<td>Determines which string argument from ErrRaise or ErrLog that should be used in this arg tag. For example 1 means that the first string argument is used.</td>
</tr>
</tbody>
</table>

Continues on next page
Example of the .xml file

This .xml file `<Add-In name>_elogtext.xml` contains the text for an error message that will look similar to this:

```xml
<?xml version="1.0" encoding="utf-8"?>
<!--The text description file for Elog Messages -->
<Domain elogDomain="PROC" domainNo="11" lang="en" elogTextVersion="1.0" xmlns="urn:abb-robotics-elog-text"
   min="5001" max="5001">
  <Message number="5001" eDefine="ERR_ARG_TO_SMALL">
    <Title>Too small value on argument</Title>
    <Description>
      Task: T_ROB1  
      The argument Radius was set to 1.00 but the minimum allowed value is 2.
      Context: /MainModule/main/MoveCircle/4
    </Description>
  </Message>
</Domain>
```

Details

Task: T_ROB1  
The argument Radius was set to 1.00 but the minimum allowed value is 2.  
Context: /MainModule/main/MoveCircle/4
2.1.3 Validating event log .xml files

Introduction

A validation tool checks that the event log .xml file is correctly formatted, using the corresponding XML schema file, elogtext.xsd.

- The schema file (elogtext.xsd) and the file template_elogtest.xml are available in the RobotWare package folder, see Template file on page 19.
- The command line tool XMLFileValidator can be downloaded from the RobotStudio Online Community, where it is included in the Tools and Utilities package.

To run the validation, start the tool and use your search paths using the principle below:

xmlfilevalidator elogtext.xsd my_elogtext.xml

The result of the validation is displayed in the console. Detailed error information including row- and column references, is displayed for any found formatting errors.

Prerequisites

The XMLFileValidator is provided as-is.

Microsoft .NET framework version 2.0 or later is required.
2.1.4 Configure event logs to take focus on the FlexPendant

Overview

Normally, new event log messages are only indicated by a notification in the FlexPendant status bar. The event log message list can then be accessed by clicking Event log in the status bar:

It is, however, possible to configure rules that enable selected event log messages to get more attention on the FlexPendant. If such rules are defined, the selected event log messages will pop up immediately on the FlexPendant screen:

Configuration setup and prerequisites

In order for the event log message configuration to work, the following must be set up:

- The messages that should get focus must be defined in the .xml file for the specific event log domain. See Create event log message rules on page 24.
- The .xml file that defines the rules must be registered for the event log domains in the install.cmd file. See Register rules in install.cmd file on page 24.
Create event log message rules

The event log message rules are defined in an .xml file.

The message numbers specified in the .xml file define what event log messages will take focus on the FlexPendant.

Note

If messages from several domains are to be configured, one .xml file must be created for each domain. Define domain with domainNo.

Example

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<rulesdomainNo="11"xmlns="urn:abb-robotics-elog-rules">
<!--Eventlog message rules for the FlexPendant -->
<message number="5001"/>
</rules>
```

Register rules in install.cmd file

The rules for each of the event log domains on the controller must be registered in the install.cmd file, using the command register.

For more information about the command register, see The install.cmd file on page 44.

Example

```
# Register event log rules for Add-In
register -type elogrules -prepath $BOOTPATH -postpath CircleMove_elogrules.xml
```
2.2 System parameters related to add-in development

2.2.1 About cfg files

Overview

The cfg files are used to define instances of system parameter types in a specific domain. The specified instances are then created by loading the cfg file. Only one domain can be specified per cfg file.

The file shall be formatted according to the rules in the following sections.

Domain specifier

A cfg file must start with a name of a domain where the specified instances will be created.

The row must contain the following information, where <version> and <revision> are optional:

```plaintext
<domain name>:CFG_1.0:<version>:<revision>::
```

Example

- Domain EIO without version number
  ```plaintext
  EIO:CFG_1.0::
  ```
- Domain EIO with version number 5.0
  ```plaintext
  EIO:CFG_1.0:5:0::
  ```
- Domain EIO with version number 6.0
  ```plaintext
  EIO:CFG_1.0:6:0::
  ```

Comments

A comment row starts with '#'.

Type specifiers

The domain specifier is followed by one or more parameter type specifiers and their instances.

- A type specifier should always be preceded by a row containing a single character '#'. (Not mandatory)
- A type specifier consists of a parameter type name directly followed by a ':'.
- There should be an empty row between the type name and the first instance. (Not mandatory)
- There should be no more rows after the last instance row in a cfg file. (Not mandatory)
- Add a description of all attributes in a type directly after the type specifier. This is helpful for the user to understand the type. (Not mandatory)

See cfg file examples later in this section.

Instances and attributes

The type specifier is followed by zero or more instances. Each instance contains one or several attributes defining its properties. Attributes can be mandatory or optional.

Continues on next page
Mandatory attributes must be specified explicitly in the cfg file otherwise an error will be generated when loading the file. Optional attributes that are not specified in the cfg file will be set to the default value for this attribute at loading. If the value of the optional attribute is specified, then the specified value will be used.

Each instance shall start with the Name attribute (if the instance has a name). Each attribute shall start with '-' (dash) followed by the attribute name, a blank space and value. Blank spaces are not allowed in the value except for string values with quotation marks.

Example:
```
-name MoveCircle -param_nr 6
```

Quotation marks can be used for string values. Note, all characters (including spaces) inside the quotation marks will be treated as one single string.

Example:
```
-name "M.C 1" -type "MMC_MC1"
```

**Single or multiple rows**

All attributes and their values in an instance can be put in a single row or in multiple rows. Comments or empty rows are not allowed in an instance. Several attributes per row are allowed.

For instances with multiple rows, each row in an instance shall end with '\\' (backslash), except for the last row. The name and the value of an attribute cannot be separated by '\\', that is, they must be on the same row.

For example, the following is not valid:
```
-name \\
"M.C 1"
```

**Arrays**

If an attribute is of an array type, then the attribute value may consist of several comma separated values. Blank spaces and the multiple row separator '\\' cannot be used inside the array.

Example:
```
-name MoveCircle -default_struct 1,1,1,1,1,0
```

**Attribute of type Boolean**

If the attribute is of type Boolean, giving only the attribute name in the cfg file will set the value to true.

Example:
```
:hidden
```

**Example of cfg file**

```
SIO:CFG_1.0::
#
COM_PHY_CHANNEL:

-Name "LAN1" -Connector "LAN"
#
COM_TRP:
# -Name Name of transmissions protocol (MAN)
```

Continues on next page
# -Type Name of transmissions protocol type (MAN)
# -PhyChannel Name of the physical channel (MAN)
# -HostName Name of host (OPT)
# -RemoteAdress Remote address (OPT)
# -Gateway Default gateway (OPT)
# -SubnetMask SubNetmask (OPT)

-Name "TCP1P1" -Type "TCP/IP" -PhyChannel "LAN1"
2.2.2 Topic Controller

About the topic Controller

This section describes system parameters that belong to the topic Controller (that is, in the configuration file sys.cfg) and that are closely related to add-in development.

The configuration of which program modules to load is made in the topic Controller. All files containing the RAPID code for the add-in must be defined here.

For more information about the types and parameters of the Controller topic, see Technical reference manual - System parameters.

Automatic loading of modules (CAB_TASK_MODULES)

The type CAB_TASK_MODULES is used to define modules to be loaded when the controller is started.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>The name of the file including the path on the controller. An environment variable can preferably be used. That is, <code>&lt;environment variable&gt;:/&lt;file name&gt;</code>. See setenv on page 58.</td>
</tr>
<tr>
<td>Task</td>
<td>Name of a task, if it should only be loaded to one specific task.</td>
</tr>
<tr>
<td></td>
<td>Note</td>
</tr>
<tr>
<td></td>
<td>The parameters Task, Shared, AllTask and AllMotionTask are mutually exclusive.</td>
</tr>
<tr>
<td>Shared</td>
<td>Defines if the contents of a module should be reachable from all tasks. The module is not loaded, it is installed, but reachable from all tasks.</td>
</tr>
<tr>
<td></td>
<td>Note</td>
</tr>
<tr>
<td></td>
<td>The parameters Task, Shared, AllTask and AllMotionTask are mutually exclusive.</td>
</tr>
<tr>
<td></td>
<td>Note</td>
</tr>
<tr>
<td></td>
<td>The parameter Shared cannot be combined with Installed.</td>
</tr>
<tr>
<td>AllTask</td>
<td>Defines if the module should be loaded into all tasks.</td>
</tr>
<tr>
<td></td>
<td>Note</td>
</tr>
<tr>
<td></td>
<td>The parameters Task, Shared, AllTask and AllMotionTask are mutually exclusive.</td>
</tr>
<tr>
<td>AllMotionTask</td>
<td>Defines if the module should be loaded into all motion tasks.</td>
</tr>
<tr>
<td></td>
<td>Note</td>
</tr>
<tr>
<td></td>
<td>The parameters Task, Shared, AllTask and AllMotionTask are mutually exclusive.</td>
</tr>
</tbody>
</table>
A module can be loaded or installed. 
A loaded module will behave like a module manually loaded from
the teach pendant.
An installed module will behave like a built in module. By default the
attributes NOVIEW and NOSTEPIN are set, even if not stated in the
module declaration. Thus it will not be visible from the FlexPendant
and can only be removed by using the restart mode Reset system.
It will not be possible to step into a routine in such a module with
FWD.
It is recommended that all application modules are installed as built
in modules, since then they will be handled as part of the controller
and quite separated from the user’s modules.

Note
The parameter Installed cannot be combined with Shared.

Hidden
RAPID routines and data in this module are hidden from the user.

Example

CAB_TASK_MODULES:
-File "CIRCLEMOVE:/CircleMove.sys" -Installed -AllTask

Modules included in a backup

There are some rules that apply when RAPID modules are saved by RobotWare
in a system backup, that add-in developers need to be aware of.

The rules are the following:

• RAPID modules that are installed or loaded directly from their add-in
  installation location (for example, using environment variables) are never
  included in a backup
• RAPID modules from location other than the product installation location
  (such as HOME):
  - If loaded – they are always included in a backup
  - If installed (including shared) - inclusion in backup depends on how
    the CFG file (CAB_TASK_MODULES instances) is loaded
      # config -internal -> not in backup
      # config (without -internal) -> included in backup

Exclude files and directories at backup

By default all files and directories in the HOME directory are included in the backup.
It is possible to exclude HOME directory files and directories from the backup. It
is also possible to include files or directories to the backup that are not located in
the HOME directory.

The text must be edited directly in the SYS.CFG file for type BACKUP_RESTORE.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExcludeFileFromHomeAtBackup</td>
<td>This file in the HOME directory shall not be included in the backup.</td>
</tr>
</tbody>
</table>
### 2 Reference material

#### 2.2.2 Topic Controller

Continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExcludeDirFromHomeAtBackup</td>
<td>This directory in the HOME directory shall not be included in the backup.</td>
</tr>
<tr>
<td>IncludeFileAtBackup</td>
<td>This file is not located in the HOME directory, but shall be included in the BACKINFO directory in the backup.</td>
</tr>
<tr>
<td>IncludeDirAtBackup</td>
<td>This directory is not located in the HOME directory, but shall be included in the BACKINFO directory in the backup.</td>
</tr>
</tbody>
</table>

**Example**

```
BACKUP_RESTORE:
-ExcludeDirFromHomeAtBackup "SecretDirectory"
-IncludeFileAtBackup "DATA:/ImportantFile.xml"
```

**Note**

The main HOME and DATA directory is intended for use by the end user RAPID program and user files.

In RobotWare 7, each add-in has its own dedicated HOME and DATA directory under the AddInData location that is separated from the main HOME and DATA directory. For more information see *Introduction on page 44*. 
2.2.3 Topic I/O System

**About the topic I/O System**

This section describes system parameters that belong to the topic *I/O System* (that is, in the configuration file *eio.cfg*).

For more information about the types and parameters of the *I/O System* topic, see *Technical reference manual - System parameters*.

**Hiding I/O signals to the user**

Add-ins can use virtual signals for internal communication, for example to communicate between RAPID tasks. It is possible to hide such signals from browsing by setting the *Access* property, for each signal, to *internal*.

It is possible to modify a hidden signal from RAPID, if the name of the signal is known and if the category of the signal is set to *RAPID*.

**Example**

```plaintext
EIO:CFG_1.0::
#
EIO_SIGNAL:
-Name "DOAccessInternal" -SignalType "DO" -Access "internal"
-Name "DOAccessInternalRAPID" -SignalType "DO" -Access "internal"
-Category "rapid"
```
2.2.4 Topic Man-machine Communication

About the topic Man-machine Communication

This section describes some of the types and system parameters in the topic Man-machine communication (that is, the configuration file mmc.cfg). It is used to define how a self-developed instruction should be presented on the FlexPendant, for example which menu to select it from (pick lists) and which argument values should be used as default (RAPID rules).

A short example is given for each type, and an example of an entire cfg file is shown after the type descriptions.

Pick list titles (MMC_PALETTE_HEAD)

It is possible to add custom pick lists alongside with the predefined pick lists that are included by default. The title for each custom pick list is defined in the MMC_PALETTE_HEAD type.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The title of the custom pick list.</td>
</tr>
<tr>
<td>type</td>
<td>The type that contains the instruction names of the pick list</td>
</tr>
</tbody>
</table>

Example

MMC_PALETTE_HEAD:
- name "M.C 1" -type "MMC_MC1"
- name "SpotWelding" -type "MMC_SPOTWELD"

Custom pick lists (MMC_MC1, MMC_MC2, MMC_MC3, etc.)

For each custom pick list there shall be an alias type definition to configure which instructions will be present in the pick list.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the instruction.</td>
</tr>
</tbody>
</table>

Note

- The pick list types contain more parameters and more functionality. For more information about these, see section Most Common Instruction Types in Technical reference manual - System parameters.
- Note the use of the equal sign to define the alias type, where the type name defined in MMC_PALETTE_HEAD is defined as an alias of the base type MMC_PALETTE.

Example

MMC_MC1 = MMC_PALETTE:
- name MoveCircle

MMC_SPOTWELD = MMC_PALETTE:
- name "SpotL"
- name "SpotJ"

Continues on next page
Default arguments (MMC_REAL_ROUTINE)

MMC_REAL_ROUTINE is used to define which arguments should have proposed values, that is, a default value when the instruction is added on the FlexPendant.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The instruction name.</td>
</tr>
<tr>
<td>default_struct</td>
<td>Defines which arguments should have proposed values.</td>
</tr>
<tr>
<td>hidden</td>
<td>Defines if the instruction should be hidden when showing RAPID routines. If hidden is set, the instruction will not be shown when choosing an instance for ProcCall or Move PP to Routine. For changes of the hidden parameter to take effect, the controller must be restarted by using the restart mode Reset RAPID or Reset system. A restart is not enough.</td>
</tr>
</tbody>
</table>

Tip

It is not necessary to specify default_struct if there should only be proposed values for required arguments.

Example

The instruction TriggInt is defined with the following arguments:

TriggInt TriggData Distance [\Start] | [\Time] Interrupt

<table>
<thead>
<tr>
<th>Argument</th>
<th>Argument number</th>
<th>Argument alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>TriggData</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Distance</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Start</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Time</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Interrupt</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that Start and Time are alternative arguments and therefore have the same argument number.

The following alternatives are examples of how to configure an instance of the type MMC_REAL_ROUTINE:

**Proposed values for TriggData, Distance, and Interrupt (the same result as if default_struct is not defined):**

- name TriggInt -default_struct 1,1,0,1

**Proposed values for TriggData, Distance, Start, and Interrupt:**

- name TriggInt -default_struct 1,1,1,1
Proposed values for TriggData, Distance, Time, and Interrupt:
- name TriggInt -default_struct 1,1,2,1

Argument reuse (MMC_INST_NOT_REUSING_PREV_OPT_ARG)
The proposed value of an instruction argument can be the same as (or in sequence with) the same argument for a previous instruction. For example, if a work object has been used in the previous move instruction, the same work object is proposed when a new move instruction is added.

If the reusing of argument values is not desired for some arguments, those arguments are specified in the type MMC_INST_NOT_REUSING_PREV_OPT_ARG.

Even if default_struct in the type MMC_REAL_ROUTINE is set to 0, an argument used in the previous instruction will be used in the next instruction. To avoid this, these arguments must also be specified in MMC_INST_NOT_REUSING_PREV_OPT_ARG.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>param_nr</td>
<td>Specifies the argument numbers that should not reuse values from previous instruction calls.</td>
</tr>
</tbody>
</table>

Example

The instruction MoveL is defined with the following arguments:

```plaintext
```

As the arguments Conc, V, T, Z, and Inpos should not be reused, the instance of MMC_INST_NOT_REUSING_PREV_OPT_ARG would look like this:

```plaintext
MMC_INST_NOT_REUSING_PREV_OPT_ARG:
-name MoveL -param_nr 1,5,7,8
```

Note that both V and T have argument number 5, as they are alternative arguments.

Argument Name Rules (MMC_REAL_PARAM)
The type MMC_REAL_PARAM is used to specify how to generate the proposed identifier for instruction arguments.

Even arguments that have default_struct in MMC_REAL_ROUTINE set to 0 and are defined in param_nr in MMC_INST_NOT_REUSING_PREV_OPT_ARG may need to be defined in MMC_REAL_PARAM. No argument proposal will be used when the instruction is chosen from a pick list, but if the argument is actively selected it will use the identifier specified in MMC_REAL_PARAM.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The instruction argument, defined as &lt;instruction name&gt;<em>&lt;argument name&gt; (for example MoveL_Tool). It is also possible to define a common argument name (common</em>&lt;argument name&gt;) to be used in the type MMC_COMMON_PARAM.</td>
</tr>
</tbody>
</table>
Parameter | Description
--- | ---
name_rule | Specifies how the argument proposal should be generated. The following rules can be used:
• NONE - Unexpanded placeholder. No proposal is generated.
• CUR - The parameter method is used to define the argument proposal. For example used when the tool argument should use the current tool.
• DEF - The argument proposal should be a default value defined by the parameter def_name.
• SEQ - The argument proposal is based on the previous instruction with a similar argument. Based on the identifier used in the previous instruction, an increment of the index is used to create a new identifier. For example, if the robtarget of the previous move instruction is p10, the next move instruction will propose p20 (unless p20 is already used, then p30, p40, ... will be tried until an identifier is found that is not already used). If no similar argument is found, looking 100 instructions back, a data value is used instead of an identifier.
• LAST - The argument proposal gets its value from the previous instruction with a similar argument. If no similar argument is found, looking 100 instructions back, a default value specified by def_name is used.
• VAL - No argument identifier is used. A literal value is used instead.
method | Method to be called if name_rule is CUR or SEQ. Supported methods are:
• hirule_robtarget - robtarget symbol name increment value
• hirule_jointtarget - jointtarget symbol name increment value
• hirule_tooldata - current tooldata
• hirule_wobjdata - current wobjdata
• hirule_tloaddata - current tload
def_name | Default name needed if name_rule is LAST or DEF.

Note
A string must have 3 quotation marks:
```
-name Direction -name_rule LAST -def_name """"Z""
```

Example

This example shows how some arguments for the MoveL instruction are configured. It also defines the common arguments common_point, common_speed, and common_zone, that are used in the type MMC_COMMON_PARAM.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Argument proposal</th>
</tr>
</thead>
</table>
| V | If V is actively selected it should:
  1. use the value from the last instruction using V
  2. use the default value 1000 |
| ID | No identifier should be proposed for ID. A numeric value is proposed instead. The proposed numeric value is defined in MMC_REAL_DATATYPE. |
| T | If T is actively selected it should use the default value 5. |
| Z | If Z is actively selected it should:
  1. use the value from the last instruction using Z
  2. use the default value 50 |
| Tool | The proposal for Tool should be defined by the method hirule_tooldata. |
| WObj | The proposal for WObj should be defined by the method hirule_wobjdata. |
## 2 Reference material

### 2.2.4 Topic Man-machine Communication

**Continued**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Argument proposal</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLoad</td>
<td>The proposal for TLoad should be defined by the method hirule_tloaddata.</td>
</tr>
</tbody>
</table>
| common_point | The proposal for common_point should:  
1. be a sequential increase from the last robtarget  
2. be defined by the method hirule_robtarget |
| common_speed | The proposal for Tool should:  
1. use the value from the last instruction using speeddata  
2. use the default value 1000 |
| common_zone | The proposal for common_zone should:  
1. use the value from the last instruction using zonedata  
2. use the default value z50 |

**MMC_REAL_PARAM:**
- name MoveL_V -name_rule LAST -def_name 1000  
- name MoveL_ID -name_rule VAL  
- name MoveL_T -name_rule DEF -def_name 5  
- name MoveL_Z -name_rule LAST -def_name 50  
- name MoveL_Tool -name_rule CUR -method hirule_tooldata  
- name MoveL_WObj -name_rule CUR -method hirule_wobjdata  
- name MoveL_TLoad -name_rule CUR -method hirule_tloaddata  
- name common_point -name_rule SEQ -method hirule_robtarget  
- name common_speed -name_rule LAST -def_name v1000  
- name common_zone -name_rule LAST -def_name z50

**Argument Identifier Rules (MMC_COMMON_PARAM)**

With the type **MMC_COMMON_PARAM**, a common argument (defined in **MMC_REAL_PARAM**) is used to define an argument proposal.

For example, a common argument defining proposals for all **ToPoint** arguments can be defined in **MMC_REAL_PARAM**. In **MMC_COMMON_PARAM**, the **ToPoint** argument for all move instructions can use that common argument.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The instruction argument, defined as <code>&lt;instruction name&gt;_&lt;argument name&gt;</code> (for example MoveL_Tool).</td>
</tr>
<tr>
<td>common_space_name</td>
<td>Name of the common argument defined in <strong>MMC_REAL_PARAM</strong>.</td>
</tr>
</tbody>
</table>

**Example**

In this example the argument proposals for the **MoveL arguments** **ToPoint**, **Speed**, and **Zone** are defined by **common_point**, **common_speed**, and **common_zone**.

**MMC_COMMON_PARAM:**
- name MoveL_ToPoint -common_space_name common_point  
- name MoveL_Speed -common_space_name common_speed  
- name MoveL_Zone -common_space_name common_zone

**Continues on next page**
Data Value Rules (MMC_REAL_DATATYPE)

The type MMC_REAL_DATATYPE is used to specify how to generate the proposed value for a data type.

When an instruction is added, the proposed argument identifiers are defined in MMC_REAL_PARAM, while the values of those arguments are defined in MMC_REAL_DATATYPE.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the data type.</td>
</tr>
<tr>
<td>def_name</td>
<td>Default base identifier for the data (for example tool). The identifier for the data is created from the def_name and an index. If nothing else is defined, the index starts at 1 and the increment for each data is 1 (for example the first tooldata is called tool1, the second is called tool2 and so on).</td>
</tr>
<tr>
<td>value_rule</td>
<td>Specifies how the value of the new data should be generated:</td>
</tr>
<tr>
<td></td>
<td>• NONE - No initialize value for non-value data type.</td>
</tr>
<tr>
<td></td>
<td>• CUR - The parameter method is used to define the data value. For example used when a robtarget is given the value of the current robot TCP.</td>
</tr>
<tr>
<td></td>
<td>• DEF - The data value should be a default value defined by the parameter use_value.</td>
</tr>
<tr>
<td></td>
<td>• SEQ - The data value is based on the previous data of the same data type. The previous value is increased with a value defined by use_value. If no data is found, when looking up to 100 statements back, a zero value is used.</td>
</tr>
<tr>
<td>method</td>
<td>Method to be called if value_rule is CUR. Supported methods are:</td>
</tr>
<tr>
<td></td>
<td>• hirule_robtarget - current robot TCP robtarget value</td>
</tr>
<tr>
<td></td>
<td>• hirule_jointtarget - current robot TCP jointtarget value</td>
</tr>
<tr>
<td></td>
<td>• hirule_tooldata - current tooldata value</td>
</tr>
<tr>
<td></td>
<td>• hirule_wobjdata - current wobjdata value</td>
</tr>
<tr>
<td></td>
<td>• hirule_toaddata - current toaddata value</td>
</tr>
<tr>
<td>use_value</td>
<td>Default value if value_rule is DEF or SEQ. Also used as increment value if value_rule is SEQ.</td>
</tr>
<tr>
<td>object_type</td>
<td>Data object type (i.e. CONST, VAR, PERS or TASK PERS).</td>
</tr>
<tr>
<td>validate_hook</td>
<td>Method to be called when validating data. Supported methods are:</td>
</tr>
<tr>
<td></td>
<td>• hirule_validate_tooldata</td>
</tr>
<tr>
<td></td>
<td>• hirule_validate_wobjdata</td>
</tr>
<tr>
<td></td>
<td>• hirule_validate_robtarget</td>
</tr>
<tr>
<td></td>
<td>• hirule_validate_orient</td>
</tr>
<tr>
<td></td>
<td>• hirule_validate_pose</td>
</tr>
<tr>
<td></td>
<td>• hirule_validate_progdisp</td>
</tr>
<tr>
<td></td>
<td>• hirule_validate_toaddata</td>
</tr>
</tbody>
</table>

Example

This example defines the proposed values for the data types identno and robtarget.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Proposed data value</th>
</tr>
</thead>
<tbody>
<tr>
<td>identno</td>
<td>If no identno exists, the value is 10. Otherwise the value from the last identno is increased with 10.</td>
</tr>
</tbody>
</table>
Proposed data value

<table>
<thead>
<tr>
<th>Data type</th>
<th>Proposed data value</th>
</tr>
</thead>
<tbody>
<tr>
<td>robject</td>
<td>The new robject gets the value of the current robot TCP. A validation is used so that the value of a robject cannot be changed to an incorrect format.</td>
</tr>
</tbody>
</table>

```
MMC_REAL_DATATYPE:
-name identno -def_name id -value_rule SEQ -use_value 10 \
-object_type CONST
-name robject -def_name p -value_rule CUR \
-method hirule_robject -object_type CONST\n-validate_hook hirule_validate_robject
```

Highlight argument (MMC_SELECT_PARAM)

When an instruction is added, one of the arguments can be automatically selected for further definitions. This is defined in the type MMC_SELECT_PARAM. For example, when adding a MoveC instruction, the CirPoint is set to the current TCP value and the ToPoint is selected for the required modify position.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>param_nr</td>
<td>Parameter number for the argument to be selected.</td>
</tr>
</tbody>
</table>

Example

The instruction MoveC is defined with the following arguments:

```
```

Since a modify position of ToPoint is required after the instruction is added, the argument ToPoint is selected:

```
MMC_SELECT_PARAM:
-name MoveC -param_nr 3
```

Work objects (MMC_INSTR_WITH_WOBJ)

MMC_INSTR_WITH_WOBJ is used when adding instructions from the FlexPendant, for which no default arguments are specified in MMC_REAL_PARAM.

It checks if the instruction has a \WObj optional argument, and what position the optional argument has in the instruction. If the active work object on the FlexPendant differs from the default work object, wobj0, then the optional argument \WObj in the instruction is added and set to the active work object.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the instruction.</td>
</tr>
<tr>
<td>param_nr</td>
<td>Argument number for the \WObj optional argument.</td>
</tr>
</tbody>
</table>

Example

```
MMC_INSTR_WITH_WOBJ:
-name MoveL -param_nr 10
```
Load objects (MMC_INSTR_WITH_TLOAD)

MMC_INSTR_WITH_TLOAD is used when adding instructions from the FlexPendant, for which no default arguments are specified in MMC_REAL_PARAM.

It checks if the instruction has a \TLoad optional argument, and what position the optional argument has in the instruction. If the active payload on the FlexPendant differs from the default payload, load0, then the optional argument \TLoad in the instruction is added and set to the active payload.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the instruction.</td>
</tr>
<tr>
<td>param_nr</td>
<td>Argument number for the \TLoad optional argument.</td>
</tr>
</tbody>
</table>

Example

```
MMC_INSTR_WITH_TLOAD:
   -name MoveL -param_nr 12
```

Circular points (MMC_INSTR_WITH_CIR_POINT)

MMC_INSTR_WITH_CIR_POINT is used for instructions with circular points, CirPoint.

After a position is modified, the controller tries to update the planned path to use the new position. This functionality needs to know if a target is a circular point.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the instruction.</td>
</tr>
<tr>
<td>param_nr</td>
<td>Argument number for the circular point, CirPoint.</td>
</tr>
</tbody>
</table>

Example

```
MMC_INSTR_WITH_CIR_POINT:
   -name MoveC -param_nr 2
```

Arguments not available for modify position (MMC_NO_MODPOS)

MMC_NO_MODPOS defines instruction arguments that should not be modified with modify position, even though they are of data type robtarget or jointtarget.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The instruction argument, defined as &lt;instruction name&gt;_&lt;argument name&gt; (for example MoveL_Tool).</td>
</tr>
</tbody>
</table>

Example

The instruction MToolTCP Calib is defined with the following arguments:

```
MToolTCP Calib Pos1 Pos2 Pos3 Pos 4 Tool MaxErr MeanErr
```

Pos1, Pos2, Pos3, Pos4 are of type jointtarget but should not be available for modify position:

```
MMC_NO_MODPOS:
   -name MToolTCP Calib_Pos1
   -name MToolTCP Calib_Pos2
   -name MToolTCP Calib_Pos3
   -name MToolTCP Calib_Pos4
```
2 Reference material

2.2.4 Topic Man-machine Communication

Continued

Targets not available for modify position when additional axes offset is active (MMC_NO_DATA_MODPOS_IF_ACT_EOFFS)

**MMC_NO_DATA_MODPOS_IF_ACT_EOFFS** defines data types, targets, that should not be modified with modify position by url (e.g. from Program Data view on the FlexPendant) if an additional axes offset is active.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the data type.</td>
</tr>
</tbody>
</table>

**Example**

```bash
MMC_NO_DATA_MODPOS_IF_ACT_EOFFS:
-name jointtarget
```

Optional argument for considering additional axes offset (MMC_USE_ACT_EOFFS_IN_MODPOS)

**MMC_USE_ACT_EOFFS_IN_MODPOS** is used to define instructions with optional arguments, that controls if an active additional axes offset shall be considered or not, when calculating the current position.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the instruction.</td>
</tr>
<tr>
<td>param_nr</td>
<td>Identifies the optional argument.</td>
</tr>
<tr>
<td>use_if_present</td>
<td>Defines if the offset shall be considered if the argument is present (1) or when it is not present (0).</td>
</tr>
</tbody>
</table>

**Example**

```bash
MMC_USE_ACT_EOFFS_IN_MODPOS:
-name MoveAbsJ -param_nr 4 -use_if_present 0
```

Between points (MMC_NO_PC_MOVEMENT)

For instructions with between point, such as MoveC, the program pointer should not continue to the next instruction after modify position of the between point. The type **MMC_NO_PC_MOVEMENT** is used to define the between points for which a modify position will not move the program pointer to the next instruction.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The instruction argument, defined as <code>&lt;instruction name&gt;_&lt;argument name&gt;</code> (for example MoveC_CirPoint).</td>
</tr>
</tbody>
</table>

**Example**

```bash
MMC_NO_PC_MOVEMENT:
-name movec_cirpoint
```
Without between point (MMC_NO_PC_MOVEMENT_CLEAR_PATH)

For instructions without between point, such as SpotL, the program pointer should not continue to the next instruction and a clear path is performed after modify position. The type MMC_NO_PC_MOVEMENT_CLEAR_PATH is used default in Spot systems to avoid disturbing event log messages and regain dialogs after modifying position.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The instruction argument, defined as <code>&lt;instruction name&gt;_&lt;argument name&gt;</code>.</td>
</tr>
</tbody>
</table>

Example

MMC_NO_PC_MOVEMENT_CLEAR_PATH:
- name SpotL_ToPoint
- name SpotJ_ToPoint
- name SpotML_ToPoint
- name SpotMJ_ToPoint

Service routines (MMC_SERV_ROUT_STRUCT)

MMC_SERV_ROUT_STRUCT is used to specify instructions that should be defined as service routines.

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

Example

In this example the instruction LoadIdentify is defined as a service routine:

MMC_SERV_ROUT_STRUCT:
- name LoadIdentify

Change of motion mode (MMC_CHANGE_MOTION_MODE)

For some move instructions it is possible to change motion mode (for example from MoveL and MoveJ). Which instructions allow change of mode and what instruction it is changed to is defined in MMC_CHANGE_MOTION_MODE.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the existing instruction.</td>
</tr>
<tr>
<td>shift_name</td>
<td>Name of the instruction it should be changed to.</td>
</tr>
<tr>
<td>shift_mode</td>
<td>Motion mode of instruction after changing motion mode.</td>
</tr>
<tr>
<td>param_restr</td>
<td>Defines an argument number. If this argument is set, change of motion is not allowed.</td>
</tr>
</tbody>
</table>

Example

This example specifies that the instruction MoveL can be changed into a MoveJ instruction. If the argument Corr is set this change of motion mode cannot be done.

MMC_CHANGE_MOTION_MODE:
- name MoveL -shift_name MoveJ -shift_mode Joint -param_restr 11
- name MoveJ -shift_name MoveL -shift_mode Linear
2.2.5 Example cfg files

Overview

This section contains cfg example files for the add-in Circlemove and the instruction MoveCircle.

CircleMove_sys.cfg

This example uses the environment variable CIRCLEMOVE that is defined in install.cmd, see Examples of install.cmd files on page 61.

SYS:CFG_1.0::
# Installation of RAPID routines for Add-In CircleMove
# $Revision: 1.7 $

#
CAB_TASK_MODULES:

-File "CIRCLEMOVE:/CircleMove.sys" -Install -AllTask

CircleMove_mmc.cfg

The instruction MoveCircle is defined with the following arguments:

MoveCircle pCenter Radius Speed Zone Tool [\WObj]

To define how MoveCircle should behave on the FlexPendant, the following configuration is placed in a file called CircleMove_mmc.cfg, which is added to the CircleMove add-in.

MMC:CFG_1.0::
# MMC : RAPID PROGRAMMING RULES FOR MODULE CIRCLEMOVE
# $Revision: 1.7 $

#
MMC_MC1 = MMC_PALETTE:

- name MoveCircle

#
MMC_REAL_ROUTINE:

- name MoveCircle -default_struct 1,1,1,1,1,0 -hidden

#
MMC_REAL_PARAM:

- name MoveCircle_pCenter -name_rule SEQ -method hirule_robtarget
- name MoveCircle_Radius -name_rule LAST def_name 10
- name MoveCircle_Speed -name_rule LAST -def_name v1000
- name MoveCircle_Zone -name_rule LAST -def_name z50
- name MoveCircle_Tool -name_rule CUR -method hirule_tooldata
- name MoveCircle_WObj -name_rule CUR -method hirule_wobdata

#
MMC_INSTR_WITH_WOBJ:

-name MoveCircle -param_nr 6
2 Reference material

2.3 The install.cmd file

2.3.1 Introduction

Description

The script install.cmd initializes the add-in and brings it to the default state. It is executed automatically on the first startup after system installation, each time after system update (using Installation Manager) and when using Reset system (I-Start). This script installs several different resource files that are packaged with the add-in, such as configuration files or text files. This section describes syntax used by the script, behavior and arguments of the commands that can be used in this script.

Main elements and concepts used by the installation scripts are the following:

- **Comments**
  All lines beginning by # followed by space are treated as comments.
  
  For example, # A comment.

- **Labels**
  All lines beginning by # followed by some text (no space between) are labels that can be used by those commands that support label arguments.
  
  For example, #LABEL_99.

- **Empty lines**
  Empty lines are lines containing no text. Any number of empty lines can be used to increase readability.

- **Commands**
  Commands are non-empty lines that do not start by "#". Commands may use zero or more arguments. Some of the arguments may be optional. Argument names are preceded by "-", for example: -path. Argument value must follow the argument name. For Boolean arguments, the value can be TRUE or FALSE and can be omitted. Specifying a Boolean argument without its value is the same as assigning it to TRUE. Values of string arguments should be surrounded by quotes. Quotes must be used in case a string argument value contains spaces (for example, print -text "This is a message").

- **Flow control**
  Installation scripts only have basic support for controlling script command execution flow. This is accomplished by "if***" group of commands and other commands that support conditional jumping on labels, depending on command result.
  
  Installation scripts do not support loops, such as "do" or "while" loops, switch statements etc. that can be found in other programming languages.

- **One vs. multiple script files**
  In case more complex logic is required in the installation script, more than one script may be created. The install.cmd remains the "main" script, but it can execute any number of other scripts by using commands "include"
and “loop”. Include command executes another script and then returns to
the current script. State can be passed between scripts by using script
variables (see below). Loop command is used to execute another script several
times in a loop.

- **Script variables**
  Installation scripts support two types of named script variables – *string variables* (the variable value is a string) and *integer variables* (the variable value is an integer). Names of the script variables are strings prefixed by $ and the maximal length of the variable name is 20 characters. Script variables defined in add-in installation scripts are not persistent, which means that they are lost when the controller restarts. For more information and examples, see commands *setintvar on page 59* and *setstr on page 59*.

Predefined script variables are the following:

<table>
<thead>
<tr>
<th>Script variable name</th>
<th>Corresponding envvar</th>
</tr>
</thead>
<tbody>
<tr>
<td>$HOME</td>
<td>Same as environment variable HOME.</td>
</tr>
<tr>
<td>$DATA</td>
<td>Same as environment variable DATA.</td>
</tr>
<tr>
<td>$BACKUP</td>
<td>Same as environment variable BACKUP.</td>
</tr>
<tr>
<td>$RWTEMP</td>
<td>Same as environment variable TEMP.</td>
</tr>
<tr>
<td>$RAMDISK</td>
<td>Same as environment variable RAMDISK.</td>
</tr>
<tr>
<td>$BOOTPATH</td>
<td>Temporary variable that points to the installation directory of the add-in that is currently initialized. It is the root folder that contains install.cmd.</td>
</tr>
</tbody>
</table>

- **Environment variables and directory structure on the controller**
  Environment variables are persistent variables that can be used in the installation scripts, RAPID programs and clients of Robot Web Services to access controller resources in a portable and uniform manner. The following table describes predefined environment variables and their intended usage:

<table>
<thead>
<tr>
<th>Environment variable name</th>
<th>ReadOnly/Read-Write</th>
<th>Intended usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOME</td>
<td>RW</td>
<td>Store and read files from RAPID program(s) and for files explicitly saved by user. If necessary, add-ins may copy some files from their product installation directory to HOME from their install script, if those files are intended to be modified by the user or developer of the RAPID program. The HOME directory is included in backups, so add-in developers must make sure that they maintain backwards compatibility for all files placed in this directory.</td>
</tr>
<tr>
<td>DATA</td>
<td>RW</td>
<td>Directory intended for logs or similar files that should not be included in backups, and which are created from RAPID programs.</td>
</tr>
</tbody>
</table>
2 Reference material

2.3.1 Introduction

Continued

<table>
<thead>
<tr>
<th>Environment variable name</th>
<th>ReadOnly/Read-Write</th>
<th>Intended usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKUP</td>
<td>RW</td>
<td>Directory intended for storing backups.</td>
</tr>
<tr>
<td>TEMP</td>
<td>RW</td>
<td>Directory for storing temporary files. This directory is cleaned on system Reset (I-Start) and during system updates.</td>
</tr>
<tr>
<td>RAMDISK</td>
<td>RW</td>
<td>Directory located in RAM disk (non-persistent) for high-performance logging. Content is lost on each restart of the controller.</td>
</tr>
<tr>
<td>XXX.YYY.ZZZ</td>
<td>RO</td>
<td>Product installation directory for an add-in. The name of the variable is capitalized product ID (from add-in product manifest). Note that this is a read-only location and the write protection cannot be removed from the add-in installation scripts.</td>
</tr>
<tr>
<td>XXX.YYY.ZZZ_HOME</td>
<td>RW</td>
<td>Add-in specific HOME directory that is included in backups and restored when restoring a backup. Each add-in should use its own HOME directory to avoid mixing its own data with data coming from other add-ins, RAPID programs and user files.</td>
</tr>
<tr>
<td>XXX.YYY.ZZZ_DATA</td>
<td>RW</td>
<td>Add-in specific DATA directory that is included in backups (for diagnostics) but not restored when restoring a backup. Each add-in should use its own DATA directory to avoid mixing its own data with data coming from other add-ins, RAPID programs and user files.</td>
</tr>
</tbody>
</table>

In addition to the system pre-defined environment variables, add-in developers can define custom variables from their installation scripts (see `setenv on page 58`). Typical use case in RobotWare 6 was to copy the value of `$BOOTPATH` script variable into an own environment variable to be able to access the add-in installation directory from RAPID code. As shown in the above table, in RobotWare 7 there are system pre-defined environment variables for each add-in installation and runtime data directories, so this is no longer necessary.

**Note**

To make sure that your add-in will work properly in RobotWare releases 7.2 and later, make sure that you only use the locations specified in the above table.

**Note**

RobotWare 7 add-ins cannot remove write protection on installed products and modify the installation from their `install.cmd` scripts.

Continues on next page
• **Environment variables vs. script variables**
  Script variables should be used as local variables for implementing script logic. Environment variables are global and persistent until the RobotWare system is reset or uninstalled and can be used from RAPID. System predefined environment variables can be used in Robot Web Services URL-s. Script commands, that access file resources, recognize and expand both script variables and environment variables before accessing files and directories. Recommended way of accessing file resources is using environment variables from the table in the previous section, since it is a uniform way that works from RAPID and Robot Web Services as well.

**Expansion rules during variable assignment**

The following expansion rules apply when using existing variables to define new variables:

• **Script variables**
  Any number of script variables are allowed in an assigned value and all those script variables are expanded during assignment to other script variables and when assigning to environment variables.

• **Environment variables**
  At most one environment variable is allowed in an assigned value. It is
  - recognized only in the beginning of the value
  - expanded during assignment to another environment variable
  - not expanded when assigning to a script variable.

• **Script and environment variables can be combined in an assigned value at the same time.**

• **Parts of an assigned string value that do not match the name of any existing variable are left unchanged.**
2.3.2 Commands

addintvar

Increment or decrement a previously defined integer variable. A variable is defined using setintvar.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the script variable.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Value to add.</td>
<td></td>
</tr>
</tbody>
</table>

Examples:

```
setintvar -name $TEST -value 123
addintvar -name $TEST -value 5
addintvar -name $TEST -value -1
```

append

This command can be used for two different purposes:

- Append a content of one file to another file
- Generate one line - "include" command into a script to make it include another script.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>Name of an existing file or script that shall be appended or included.</td>
<td></td>
</tr>
<tr>
<td>to</td>
<td>Path of the file in which the content shall be appended to.</td>
<td></td>
</tr>
<tr>
<td>paste</td>
<td>If FALSE, then &quot;include&quot; statement shall be generated in the script specified by &quot;to&quot; argument (append by reference). If TRUE, then contents of file &quot;from&quot; shall be appended to the file &quot;to&quot; (append by value).</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Examples:

```
append -from OPEN.ABB.ROBOTICS.CIRCLEMOVE/myfile1.txt -to OPEN.ABB.ROBOTICS.CIRCLEMOVE_HOME/myfile.txt -paste
append -from OPEN.ABB.ROBOTICS.CIRCLEMOVE/myfile3.txt -to OPEN.ABB.ROBOTICS.CIRCLEMOVE_HOME/myfile.txt -paste
```

attrib

This command can be used to modify file or directory attributes, such as read-write or read-only.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of an existing file or directory that shall be modified.</td>
<td></td>
</tr>
</tbody>
</table>
| attrs     | Attributes to modify: 
- \( -R \) – remove write protection from a file or directory 
+ \( +R \) – apply write protection on a file or directory |         |

Example:

```
attrib -path OPEN.ABB.ROBOTICS.CIRCLEMOVE_HOME/myfile.txt -attrs "-R"
```
cfg_create_type_from_xml

This command can be used to create a new type in the system parameter database, based on the type description provided in the specified XML file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of XML file containing type definition.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

cfg_create_type_from_xml -path OPEN.ABB.ROBOTICS.CIRCLEMOVE/mytypes.xml

Note

This command is currently provided to ease the migration between RobotWare 6 and RobotWare 7. However, description of the XML file syntax is not published yet.

cfg_create_type_from_rules_def

This command can be used to create a new type in the system parameter database, based on the type description provided in the specified XML file which is in CFG rules format.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Path of XML file containing type definition. Types can currently only be created in the PROC domain of the configuration database.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

cfg_create_type_from_rules_def -name OPEN.ABB.ROBOTICS.CIRCLEMOVE/mytypes.xml

Note

This command is currently provided to ease the migration between RobotWare 6 and RobotWare 7. However, description of the XML file syntax is not published yet.

config

The config command can be used to load configuration resource files into the cfg object and to modify existing contents of a configuration domain.

The following operations are possible:

- Add new cfg types and instances
- Replace existing instances
- Modify attribute values for existing instances
- Write-protect instances or protect instances from deletion

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>string</td>
<td>The full cfg file name, including the path of the file to load.</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal</td>
<td>boolean</td>
<td>Write-protect the <em>cfg</em> instances, defined in the <em>cfg</em> file. The write-protected instances will not be included when <em>cfg</em> data is saved to file from the FlexPendant, RobotStudio or from any Add-in application, or be part of a backup. That is, only data that is part of the RobotWare release or an application installed on the controller should be write-protected. Internal instances are not modifiable.</td>
<td>FALSE</td>
</tr>
<tr>
<td>replace</td>
<td>boolean</td>
<td>All instances defined in the <em>cfg</em> file will replace existing instances with the same name in the <em>cfg</em> domain. To create a new instance and keep all modified values, the user has to define all attributes and their values in the <em>cfg</em> file. If any attribute is not defined in the file, the default value will be used.</td>
<td>FALSE</td>
</tr>
<tr>
<td>modify</td>
<td>boolean</td>
<td>Modify the attribute values, defined in the <em>cfg</em> file, in existing named instances. Values of other attributes which are not included in the <em>cfg</em> file will remain the same.</td>
<td>FALSE</td>
</tr>
<tr>
<td>nondeletable</td>
<td>boolean</td>
<td>Protect the <em>cfg</em> instances, defined in the <em>cfg</em> file, from deletion.</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

The boolean arguments can only be used one at a time. If more than one is used, the argument that has lower precedence will be ignored. The order of precedence is as follows:

- `nondeletable` > `modify` > `(-replace && -internal)` > `internal` > `replace`

#### Note

The only exception to the order of precedence is that the arguments `-replace` and `-internal` may be used at the same time.

### Examples

All calls of the command `config` require that the argument `-filename` is specified.

Add *cfg* types and instances without write-protection:

```
config -filename $BOOTPATH/mysys.cfg
```

Add *cfg* types and instances with write-protection:

```
config -filename $BOOTPATH/mysys.cfg -internal
```

Replace existing instances and add new instances:

```
config -filename $BOOTPATH/myeio.cfg -replace
```

Modify attribute values for named instances that are not write-protected:

```
config -filename $BOOTPATH/mymoc.cfg -modify
```
Protect the cfg instances from deletion by e.g. the command delete_cfg_instance:
```
config -filename $BOOTPATH/mymoc.cfg -nondeletable
```

Example on usage of -modify argument

This example shows how to modify the value for attribute -Devicemap from 15 to 14 for the named instance custom_DO_7.

```
Note
Before using -modify, custom_DO_7 must already exist in the domain.
```

Contents in the saved eio.cfg:
```
EIO:CFG_1.0:7:0::
#
...
#
EIO_SIGNAL:
-Name "custom_DO_7" -SignalType "DO" -Device "ManipulatorIO"
-DeviceMap "15"
```

Script command:
```
config -filename $HOME/eio_modify.cfg -modify
```

Contents in eio_modify.cfg:
```
EIO:CFG_1.0:7:0::
#
EIO_SIGNAL:
-Name "custom_DO_7" 
DeviceMap "14"
```

copy

Copy a file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>The file to be copied, including the file path.</td>
<td></td>
</tr>
<tr>
<td>to</td>
<td>The new file name, including the file path.</td>
<td></td>
</tr>
</tbody>
</table>

Example:
```
copy -from $BOOTPATH/instopt.cmd -to $RWTEMP/instopt.cmd
```

delay

Delay the running of the command script.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Number of milliseconds to delay.</td>
<td>100</td>
</tr>
</tbody>
</table>

Example:
```
delay -time 1000
```
2 Reference material

2.3.2 Commands

Continued

**delete**

Delete a file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of file to delete, including file path.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

dele -path $RWTEMP/opt_l0.cmd

**direxist**

If a directory exists, go to a label.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The complete path to the folder.</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>The label to go to if the folder exists.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

direxist -path $TEMP/MyFolder -label CLEANUP_0

**echo**

Echo (print) a message to the system console and FlexPendant system startup screen.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The text to show on the FlexPendant startup screen. This text can contain arguments such as $ANSWER and will be converted before it is displayed.</td>
<td></td>
</tr>
<tr>
<td>elog</td>
<td>Adds an internal event log with the text as message.</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Examples:

echo -text "Installing configuration files"
echo -text "Error when installing configuration for $ANSWER" -elog

**fileexist**

If a file exists, go to a label.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>File name, including the file path.</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>The label to go to if the file exists.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

fileexist -path $RWTEMP/opt_l0.cmd -label CLEANUP_0

**find_replace**

Find and replace occurrences of a string in a file. Only the first occurrence of the string in each line of the text is replaced.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>File to search, including the file path.</td>
<td></td>
</tr>
<tr>
<td>find</td>
<td>String to find.</td>
<td></td>
</tr>
<tr>
<td>replace</td>
<td>String to replace with.</td>
<td></td>
</tr>
</tbody>
</table>
getkey

A number of selections can be made by user at the time of system creation. Values of these selections come from product manifest file and are stored by the system as a number of keys. The values stored in these keys can be read at the system startup time using the `getkey` command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Name of the key whose value is to be retrieved.</td>
<td></td>
</tr>
<tr>
<td>strvar</td>
<td>Name of the variable where the result (the key value) is stored.</td>
<td></td>
</tr>
<tr>
<td>errlabel</td>
<td>Label to go to if an error occurs.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
getkey -id "LangSelect" -strvar $ANSWER -errlabel ENGLISH
```

goto

Go to a label.

The label to go to can either be specified directly, using the parameter `label`, or via a string containing the label name, using the parameter `strvar`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>strvar</td>
<td>A string containing the label name to go to.</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>Label to go to</td>
<td></td>
</tr>
</tbody>
</table>

Examples:

```
goto -strvar $ANSWER
 goto -label END_LABEL
```

if_feature_present

This command tests if the specified optional product feature is currently present in the system configuration and directs execution flow in the installation script to the specified label.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The ID of the feature that should be checked.</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>Label to go to if the feature is present.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
if_feature_present -id
 abb.robotics.robotcontrol.options.multitasking -label MULTITASKING_AVAILABLE
print -text "RAPID multitasking is not available."
goto -label NEXT_STEP
#MULTITASKING_AVAILABLE
...
#NEXT_STEP
```
ifintvar

Compares an integer variable and the specified value, and if equal jumps to the specified label. If not equal, the next statement is executed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the script variable.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Integer value to compare to.</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>Label to go to if values are equal.</td>
<td></td>
</tr>
</tbody>
</table>

Example:
```plaintext
ifintvar -name $NUMBER_OF_CYCLES -value 5 -label SELECTION_5
...
#SELECTION_5
```

ifstr

If a string variable is equal to a string value, go to the specified label. If not equal, the next statement is executed.

If the string variable is undefined, the command returns an error code.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>strvar</td>
<td>String variable to be compared with a string value.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>String value to compare the string variable with.</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>Label to go to if the comparison is true.</td>
<td></td>
</tr>
</tbody>
</table>

Example:
```plaintext
ifstr -strvar $ANSWER -value "IRT5454_2B" -label APP2
```

ifvc

If the script containing this command is run on the virtual controller, go to the specified label.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>Label to go to if the script is run on a virtual controller.</td>
<td></td>
</tr>
</tbody>
</table>

Example:
```plaintext
ifvc -label NO_START_DELAY
```

include

Include the script of another command file. Executes all commands in the script and then return to the current script.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The file name of the included script, including the file path.</td>
<td></td>
</tr>
</tbody>
</table>

Example:
```plaintext
include -path $BOOTPATH/instdrv.cmd
```

loop_break

Used to break execution of loop_include.

No parameters.
Example:
See `loop_include` on page 55.

---

**loop_include**

Used to execute a script in a loop.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path of the script to execute in loop.</td>
<td></td>
</tr>
<tr>
<td>cycles</td>
<td>Maximal number of times to execute the specified script.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
install.cmd
loop_include -path $BOOTPATH/script2.cmd -cycle 5
```

```
script2.cmd
print -text "Executing script 2"
...
loop_break
```

---

**math_lib_set_mem_size**

Used to increase the size of the memory pool used for matrix calculations in RAPID.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>The size in bytes.</td>
<td>20000 bytes</td>
</tr>
</tbody>
</table>

The default size is 20000 bytes.

- Minimum allowed size is 20000 (same as default size).
- Maximum allowed size is 20000000, that is, 20 MB.

If several calls to `math_lib_set_mem_size` are made, the largest value is used.

---

**mkdir**

Make a directory.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Directory name, including the path.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
mkdir -path $RNTEMP/newdir
```
**onerror**

Set the default behavior of the script motor in case a script command fails and returns an error status code.

It is always the most recent `onerror` command that sets the current default behavior. The `onerror` semantics of included scripts does not affect the `onerror` semantics of any script that includes it.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
</table>
| action    | Defines if an error should result in: go to label, continue execution, stop execution, system failure or return from included script to the including script. Defines what behavior an error should result in. The allowed values are:  
  - `goto` - Go to a label  
  - `continue` - Ignore errors and continue execution  
  - `stop` - Stop execution of startup task using `assert()`  
  - `sysfail` - Call `SYS_FAIL()`  
  - `return` - If used by a script included by another script, execution returns to the calling script. The included script returns an error code that needs to be handled by the including script. | continue |
| label     | The label to go to if action is `goto`. | |

**Examples:**

```plaintext
onerror -action goto -label MY_LABEL1  
onerror -action continue  
onerror -action stop  
onerror -action sysfail  
onerror -action return
```

**print**

Prints a text to the system console.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The text to show on the console.</td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

```plaintext
print -text "Copying files to $BOOTPATH"
```

**rapid_delete_palette**

Deletes a picklist in the FlexPendant programming window.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>palette</td>
<td>The name of the picklist to be deleted.</td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

```plaintext
rapid_delete_palette -palette "M.C 3"  
r rapid_delete_palette -palette "Settings"
```
**rapid_delete_palette_instruction**

Deletes a RAPID instruction in a picklist in the FlexPendant programming window.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>palette</td>
<td>The name of the picklist.</td>
<td></td>
</tr>
<tr>
<td>instruction</td>
<td>The name of the RAPID instruction to be deleted.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
rapid_delete_palette_instruction -palette "Common" -instruction "FOR"
```

```
rapid_delete_palette_instruction -palette "Common" -instruction ":="
```

```
rapid_delete_palette_instruction -palette "Common" -instruction "MoveAbsJ"
```

```
rapid_delete_palette_instruction -palette "M.C 1" -instruction "MoveJ"
```

**register**

Registers additional information from an xml to controller registers, depending on the type parameter. The supported types are:

- **Error messages (elogmes)** – register the xml-file to the `elogtext_registry.xml` file. Once registered, these messages can be used by the RAPID program.
- **Error messages rules (elogrules)** – register the xml-file to the `elogtext_registry.xml` file. Once registered, these messages will get focus on the FlexPendant screen.
- **Options (option)** - Registers the option in the `option_registry.xml` file. This will enable automatic loading of FlexPendant applications from the `WebApps` folder for the add-in.
- **RAPID meta data (rapid_metadata)** – Registers additional RAPID argument settings to the `rapid_metadata_registry.xml`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Applies to type</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Defines which type (for example elogmes, option, rapid_metadata, or rapid_text) that is being registered.</td>
<td></td>
</tr>
<tr>
<td>domain_no</td>
<td>Error messages are stored in different domains. Which domain to register in is defined by domain_no. For add-ins, domain_no should always be 9.</td>
<td>elogmes</td>
</tr>
<tr>
<td>min</td>
<td>The first message number in the file being registered.</td>
<td>elogmes</td>
</tr>
<tr>
<td>max</td>
<td>The last message number in the file being registered.</td>
<td>elogmes</td>
</tr>
<tr>
<td>prepath</td>
<td>The path to the language directory.</td>
<td>elogmes, elogrules, rapid_metadata</td>
</tr>
<tr>
<td>postpath</td>
<td>The rest of the path, after the language directory, including the character \ (backslash) and the file name.</td>
<td>elogmes, elogrules, rapid_metadata</td>
</tr>
<tr>
<td>extopt</td>
<td>A flag indicating that the add-in is an external add-in.</td>
<td>option</td>
</tr>
</tbody>
</table>

Continues on next page
2 Reference material

2.3.2 Commands

Continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Applies to type</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>The name of the add-in.</td>
<td>option</td>
</tr>
<tr>
<td>path</td>
<td>The path to the add-in.</td>
<td>option</td>
</tr>
</tbody>
</table>

Examples:

# Register event log message for Add-In
register -type elogmes -domain_no 11 -min 5001 -max 5001 -prepath $BOOTPATH/language/-postpath /CircleMove_elogtext.xml -extopt

# Register event log rules for Add-In
register -type elogrules -prepath $BOOTPATH -postpath CircleMove_elogrules.xml

# Register path for Add-In
register -type option -description MyAddIn -path $BOOTPATH

# Register path for RAPID meta data
register -type rapid_metadata -prepath $HOME/ -postpath my_rapid_edit_rules.xml
register -type rapid_text -min 1 -max 123 -resource myAddIn -prepath $BOOTPATH/language/ -postpath myAddInTexts.xml

rename

Rename a specified file or directory.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>Path of the existing original file or directory.</td>
<td></td>
</tr>
<tr>
<td>to</td>
<td>New name.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

rename -from $TEMP/myfile.txt -to $TEMP/myfile.txt.old

setenv

Define an environment variable and set its value.

An environment variable can be used in the RAPID code or in cfg files.

For more information about environment variables, see Introduction on page 44.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The environment variable to be assigned a new value.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>The string to assign to the environment variable.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

setenv -name CIRCLEMOVE -value $BOOTPATH

Continues on next page
setintvar

Define a string variable, if it is not defined and then set its long integer value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of a new or an existing script variable.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Value to set.</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
setintvar -name $COUNTER -value 10
```

setstr

Define a string variable and set its value. The string can only be used in the installation script.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>strvar</td>
<td>The string variable to be assigned a new string.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>The string to assign to the string variable.</td>
<td></td>
</tr>
</tbody>
</table>

Examples:

```
setstr -strvar $LANG -value "en"
setstr -strvar $CFGPATH -value $SYSVAR
```

text

This command loads a text description file into a text resource of a package. It accomplishes the same thing as the RAPID instruction TextTabInstall, but can also specify different texts for different languages.

For more information, read about user message functionality in Application manual - Controller software OmniCore, and Overview on page 64.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Name of the description file, including the file path.</td>
<td></td>
</tr>
<tr>
<td>package</td>
<td>Package for building the text resource.</td>
<td>&quot;en&quot;</td>
</tr>
</tbody>
</table>

Example:

```
text -filename $BOOTPATH/language/en/text_file.xml -package "en"
```

timestamp

Read the system clock and print number of seconds and milliseconds to the standard output.

No parameters.

xattrib

Extended attrib command that works recursively on a directory structure, including all subdirectories.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of a file directory to modify.</td>
<td></td>
</tr>
</tbody>
</table>
2 Reference material

2.3.2 Commands

Continued

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrs</td>
<td>Attributes to modify:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-R – remove write protection from a file or directory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+R – apply write protection on a file or directory</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
xattrib -path OPEN.ABB.ROBOTICS.CIRCLEMOVE_HOME/dir1 -attrs "-R"
```

xcopy

Recursively copy a directory structure from one location to another. It is possible to use wildcards.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>Name of a directory to copy.</td>
<td></td>
</tr>
<tr>
<td>to</td>
<td>Location to copy to.</td>
<td></td>
</tr>
<tr>
<td>create_dest_dir</td>
<td>If specified, the destination directory shall be created if it does not exist.</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Examples:

```
xcopy -from $BOOTPATH/MyDir -to $TEMP/MyDir -create_dest_dir
xcopy -from $BOOTPATH/MyDir/a*.txt -to $TEMP/MyDir_txt -create_dest_dir
```

xdelete

Recursively delete a directory structure. It is possible to use wildcards.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Name of a directory to delete.</td>
<td></td>
</tr>
<tr>
<td>unprotect</td>
<td>Remove write protection if necessary.</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Examples:

```
xdelete -path $TEMP/MyDir
xdelete -path $TEMP/MyDir2/*
```
2.3.3 Examples of install.cmd files

Example for CIRCLEMOVE

# Install.cmd script for Add-In CIRCLEMOVE
echo -text "Installing CIRCLEMOVE Add-In"
# Load configuration files
config -filename $BOOTPATH/CircleMove_sys.cfg -domain SYS -internal
config -filename $BOOTPATH/CircleMove_mmc.cfg -domain MMC
# Define environment variable
setenv -name CIRCLEMOVE -value $BOOTPATH
# Register elog messages
register -type elogmes -domain_no 11 -min 5001 -max 5001 -prepath
    $BOOTPATH/language/ -postpath /CircleMove_elogtext.xml
2 Reference material

2.4 RAPID

2.4.1 RAPID modules

Overview

The RAPID code, implementing the functionality of your add-in, is written in a system module (.sys) file (preferably <Add-In name>.sys).

Tip

By setting the argument NOSTEPIN on the module, stepwise execution of the RAPID program will not step into the module. This makes a routine written in the module behave like an instruction delivered from ABB.

RAPID code example

This is an example of how to create your own move instruction and how to use your own error messages. An instruction, MoveCircle, is created that moves the robot TCP in a circle around a robtarget, with the radius given as argument. If MoveCircle is called with a too small radius, a message defined in an .xml file is written to the event log, see Event log texts on page 20.

```
MODULE CIRCLEMOVE(SYSMODULE, NOSTEPIN)

VAR errnum ERR_CIRCLE:= -1;
VAR num errorid := 5001;

PROC MoveCircle(
    robtarget pCenter,
    num Radius,
    speeddata Speed,
    zonedata Zone,
    PERS tooldata Tool
    \PERS wobjdata WObj)

    VAR robtarget p1;
    VAR robtarget p2;
    VAR robtarget p3;
    VAR robtarget p4;

    BookErrNo ERR_CIRCLE;
    IF Radius < 2 THEN
        ErrRaise "ERR_CIRCLE", errorid, ERRSTR_TASK, "Radius",
            NumToStr(Radius,2), "2", ERRSTR_CONTEXT;
    ENDIF

    p1:=pCenter;
    p2:=pCenter;
    p3:=pCenter;
    p4:=pCenter;
```

Continues on next page
p1.trans:=pCenter.trans+[0,Radius,0];
p2.trans:=pCenter.trans+[Radius,0,0];
p3.trans:=pCenter.trans+[0,-Radius,0];
p4.trans:=pCenter.trans+[−Radius,0,0];
MoveL p1,Speed,Zone,Tool\WObj?WObj;
MoveC p2,p3,Speed,z10,Tool\WObj?WObj;
MoveC p4,p1,Speed,Zone,Tool\WObj?WObj;

BACKWARD
   MoveL p1,Speed,Zone,Tool\WObj?WObj;

ERROR
   IF ERRNO = ERR_CIRCLE THEN
      TPWrite "The radius is too small";
      RAISE;
   ENDIF

ENDPROC
ENDMODULE
2 Reference material

2.4.2 Using text resources from files

Overview

It is possible to use text strings from a text table file. This is useful, for example, when a message to the user should be displayed in different languages.

How to use text table files is described in section Advanced RAPID in Application manual - Controller software OmniCore.

Including language files from your add-in

Localized files can be installed by moving their installation to a separate install.cmd file and including it from the main installation script.

```
include -path "$BOOTPATH/language/install.cmd"
```

The add-in folder must contain a subfolder called language with a separate install.cmd file used to install the localized files. The localized files are placed in language specific subfolders of the folder language. The subfolders should be named with the 2 letter language code, for example en, de, fr etc. See illustration in section Recommended file structure on page 12.

The file install.cmd will call the file instlang.cmd in the language folder once for every installed language on the robot controller with the variable $LANG set to the corresponding language code. After this process has completed the $LANG variable will always be reset to en.

If using the RAPID instruction TextGet, place the text strings in the respective language folder in a file ending with text.xml.

Example

Example of instlang.cmd, how to install a localized file.

```
fileexist -path $BOOTPATH/language/$LANG/CircleMove_text.xml -label INSTALL_FILE

#INSTALL_FILE

goto -label END

#END

text -filename $BOOTPATH/language/$LANG/CircleMove_text.xml -package $LANG

#END
```
2.4.3 Hiding RAPID content

Overview

It is possible to hide the implementation of RAPID code on the FlexPendant. Developers of add-ins often expose a public interface to their functionality that other RAPID programmers and end users can access. It is a good programming practice to hide parts of the internal implementation that are not intended for the users of your add-in.

This section describes some recommendations for hiding the code.

Split the code into two modules

One way of hiding the code is to split the code into two modules. The first module contains the implementation that shall be hidden, and the second module contains the public interface which is visible. The interface module contents will be visible but the code can be encrypted.

For more information, see Automatic loading of modules (CAB_TASK_MODULES) on page 28.

Example

```sys.cfg
CAB_TASK_MODULES:
-File "CIRCLEMOVE:/CircleMoveImpl.sys" -Hidden -AllTask
-File "CIRCLEMOVE:/CircleMove.sys" -AllTask

CircleMove.sys - Interface

MODULE CIRCLEMOVE(SYSMODULE, NOSTEPIN)
PROC MoveCircle(
  robtarget pCenter,
  num Radius,
  speeddata Speed,
  zonedata Zone,
  PERS tooldata Tool
  \PERS wobjdata WObj)

  MoveCircleImpl pCenter, Radius, Speed, Zone, Tool \WObj?WObj;

ENDPROC
ENDMODULE
```

Continues on next page
CircleMoveImpl.sys - Implementation

```plaintext
MODULE CIRCLEMOVEIMPL(SYSMODULE, NOVIEW)
  VAR errnum ERR_CIRCLE:= -1;
  VAR num errorid := 5001;
  PROC MoveCircleImpl(
    robtarget pCenter,
    num Radius,
    speeddata Speed,
    zonedata Zone,
    PERS tooldata Tool
    \PERS wobjdata WObj)
    ...
  ENDPROC
ENDMODULE
```

Use hidden modules and the pick list

Another method is to place all code in a hidden module and use the pick list to call the procedures.

For more information, see Custom pick lists (MMC_MC1, MMC_MC2, MMC_MC3, etc.) on page 32.

Example

```plaintext
sys.cfg
  CAB_TASK_MODULES:
    -File "CIRCLEMOVE:/CircleMove.sys" -Hidden -AllTask

mmc.cfg
  MMC_CIRCLEMOVE_PALETTE = MMC_PALETTE:
    -name "MoveCircle"
  MMC_PALETTE_HEAD:
    -name "Move Circle Palette" -type "MMC_CIRCLEMOVE_PALETTE"
```
2.4.4 Optional settings for RAPID arguments (RAPID meta data)

Overview

It is possible to specify certain optional settings for arguments in RAPID instructions. For instance it is possible to define if certain arguments shall be hidden when viewing the RAPID program on the FlexPendant.

The optional settings are specified in an .xml file.

XML format

```xml
<?xml version="1.0" encoding="utf-8"?>
<Rapid>
  <Edit>
    <Instruction name="Instr1">
      <Argument name="Arg1" show="true" showeditor="false" />
    </Instruction>
  </Edit>
</Rapid>
```

Tip

Use the template file named `rapid_edit_rules.xml` located in the following directory in the RobotWare package folder:

```
...\RobotPackages\RobotWare_RPK_<version>\utility\Template\RAPID Optional Arguments
```

Navigate to the RobotWare installation folder from the RobotStudio Add-Ins tab, by right-clicking on the installed RobotWare version in the Add-Ins browser and selecting Open Package Folder.

Name and location of the .xml file

The .xml file shall be registered using the setup script (see `register on page 57`) or should be named `rapid_edit_rules.xml` and installed in the `$(HOME)` directory of the controller.

Continues on next page
2.4.4.1 Hiding arguments in programs

Overview

It is possible to hide any of the arguments listed when displaying a programmed RAPID instruction in the Program Editor and the Production Window on the FlexPendant.

Which arguments to be shown in program windows is specified in the .xml file using the showeditor attribute. The default value is that arguments shall be shown.

XML format

```xml
<?xml version="1.0" encoding="utf-8"?>
<Rapid>
  <Edit>
    <Instruction name="Instr1">
      <Argument name="Arg1" showeditor="true" />
      <Argument name="Arg2" showeditor="false" />
    </Instruction>
  </Edit>
</Rapid>
```

Example

This is an example of an .xml file specifying which optional arguments to show for MoveJ.

```xml
<?xml version="1.0" encoding="utf-8"?>
<Rapid>
  <Edit>
    <Instruction name="MoveJ">
      <Argument name="Conc" showeditor="true" />
      <Argument name="ID" showeditor="true" />
      <Argument name="V" showeditor="true" />
      <Argument name="T" showeditor="false" />
      <Argument name="Z" showeditor="false" />
      <Argument name="Inpos" showeditor="false" />
      <Argument name="WObj" showeditor="true" />
      <Argument name="TLoad" showeditor="false" />
    </Instruction>
  </Edit>
</Rapid>
```
The result will be that only the arguments Conc, ID, V and WObj will be shown in the program windows on the FlexPendant for the instruction MoveJ.

**Note**

Hiding an argument has priority over other functions such as selection of argument when adding an instruction, see *Highlight argument (MMC_SELECT_PARAM) on page 38*, or additional optional argument in pick lists, see *Pick list titles (MMC_PALETTE_HEAD) on page 32*. For the latter case the argument will be added, but it will not be shown.


2 Reference material

2.4.4.2 Hiding optional argument when changing selected instruction

Overview

It is possible to hide any of the optional arguments listed when a RAPID instruction is changed from the FlexPendant.

Which optional arguments to be shown on the FlexPendant is specified in the .xml file using the show-attribute. The default value is that arguments shall be shown.

XML format

```xml
<?xml version="1.0" encoding="utf-8"?>
<Rapid>
  <Edit>
    <Instruction name="Instr1">
      <Argument name="Arg1" show="true" />
      <Argument name="Arg2" show="false" />
    </Instruction>
  </Edit>
</Rapid>
```

Example

This is an example of an .xml file specifying which optional arguments to show for MoveJ.

```xml
<?xml version="1.0" encoding="utf-8"?>
<Rapid>
  <Edit>
    <Instruction name="MoveJ">
      <Argument name="Conc" show="true" />
      <Argument name="ID" show="true" />
      <Argument name="V" show="true" />
      <Argument name="T" show="false" />
      <Argument name="Z" show="false" />
      <Argument name="Inpos" show="false" />
      <Argument name="WObj" show="true" />
    </Instruction>
  </Edit>
</Rapid>
```

The result will be that only the optional arguments Conc, ID, V, and WObj will be shown when changing the instruction on the FlexPendant for the instruction MoveJ.

Usage

<table>
<thead>
<tr>
<th>show</th>
<th>showeditor</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;not defined&gt;</td>
<td>&lt;not defined&gt;</td>
<td>Default, same as True, True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>Shown everywhere in FP</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>Hidden in Program Editor and Production Window</td>
</tr>
</tbody>
</table>
### 2.4.4.2 Hiding optional argument when changing selected instruction

*Continued*

<table>
<thead>
<tr>
<th>show</th>
<th>showeditor</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
<td>Hidden in Argument Window, but shown in <strong>Program Editor</strong> and <strong>Production Window</strong>. Users will not be able to program arguments having this combination, thus it is unlikely that users will be exposed to this combination. Which means that in practice this is more like False/False.</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>Totally hidden, cannot be edited by <strong>Program Editor</strong></td>
</tr>
</tbody>
</table>
2 Reference material

2.4.4.3 Argument filter

Overview

It is possible to filter the data that is shown as arguments listed on the FlexPendant and in RobotStudio.

The filter for a specific parameter is specified in the .xml file using the filter-attribute. The default value is that no filter is used.

XML format

```xml
<?xml version="1.0" encoding="utf-8"?>
<Rapid>
  <Edit>
    <Instruction name="Instr1">
      <Argument name="Arg1" filter="PLC_do_.*" />
    </Instruction>
  </Edit>
</Rapid>
```

In the example above only data with a name starting with "PLC_do_" will be matched and shown for the parameter "Arg1" in the instruction "Instr1".

Regular expressions

The regular expressions are a powerful mechanism when it comes to matching a multitude of names with a single expression.

In a regular expression all alphanumeric characters match, for example the expression "abc" will match the sequence "abc". Regular expressions are case sensitive. Most other characters also match, but a small set is known as the meta-characters. These are:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Marks the beginning of the name being matched. Default.</td>
</tr>
<tr>
<td>$</td>
<td>Marks the end of the name being matched. Default.</td>
</tr>
<tr>
<td>.</td>
<td>Any single character.</td>
</tr>
<tr>
<td>[s]</td>
<td>Any character in the non-empty set s, where s is a sequence of characters. Ranges may be specified as c-c.</td>
</tr>
<tr>
<td>[^s]</td>
<td>Any character not in the set s.</td>
</tr>
<tr>
<td>r*</td>
<td>Zero or more occurrences of the regular expression r.</td>
</tr>
<tr>
<td>r+</td>
<td>One or more occurrences of the regular expression r.</td>
</tr>
<tr>
<td>r?</td>
<td>Zero or one occurrence of the regular expression r.</td>
</tr>
<tr>
<td>(r)</td>
<td>The regular expression r. Used to separate a regular expression from another.</td>
</tr>
<tr>
<td>r</td>
<td>r'</td>
</tr>
</tbody>
</table>

Continues on next page
Examples

Some examples:

- The expression "MoveL" (or "MoveLS") would match the name "MoveL", and nothing else.
- The expression "Move.*" would match "MoveL", "MoveC", "MoveCDO" etc.
- The expression ".*Move.*" would match the names "MyMove", "MoveL", "MoveC", "MoveCDO" etc.
- The expressions "", ".*", or "^.*$", i.e. an empty string, matches anything.
2.4.4.4 Argument value range

Overview

It is possible to define minimum and maximum allowed value when specifying a numerical value for an argument. The value will be validated by the FlexPendant and RobotStudio when entering such a value.

The minimum and maximum allowed values for a specific parameter is specified in the .xml file using the minvalue and maxvalue attributes. The default value is that no minimum and maximum values are used.

XML format

```xml
<?xml version="1.0" encoding="utf-8"?>
<Rapid>
  <Edit>
    <Instruction name="Instr1">
      <Argument name="Arg1" minvalue="1" maxvalue="16" />
    </Instruction>
  </Edit>
</Rapid>
```

In the example above only values between 1 and 16 will be allowed when entering a numerical value for the parameter "Arg1" in the instruction "Instr1".

Note

The check for valid numerical value will only be performed when entering a numerical value as argument. No validation will be performed if for instance a variable is used as argument.
2.5 RobotWare Add-In Packaging tool

2.5.1 Introduction

2.5.1.1 About the RobotWare Add-In Packaging tool

General

RobotWare Add-In Packaging tool (APT) is a Windows program that helps to pack the add-in as a package that can be deployed to the robot controller via the Installation Manager. The output of the RobotWare Add-In Packaging tool is a product manifest file and a robot package file.

The tool helps you to:

• Package new RobotWare 6 add-ins.
• Package new RobotWare 7 add-ins.
• Package RobotWare 6 add-ins based on RobotWare 5 additional options.
• Define how the end-user will see the add-in product in the Installation Manager.
• Define one or more optional features and rules for how options can be selected in the Installation Manager.
• Define dependencies between your add-in and other products (RobotWare and other add-ins).

The RobotWare add-in and the RobotWare add-in license can then be used together with RobotWare to create a RobotWare system using the Installation Manager in RobotStudio.

For more information about the Installation Manager, see Operating manual - RobotStudio.

Tip

See also the tutorials on using the RobotWare Add-In Packaging tool available at ABB Library Download Center.

Open and licensed add-ins

There are two major types of add-ins that can be created with the RobotWare Add-In Packaging tool, open add-ins and licensed add-ins.

For open add-ins, the product manifest and the robot package file created will contain everything required for the user to install the product unsigned.

For licensed add-ins, there is also a signing step involved in the packaging of the add-ins, that will later allow you to generate licenses for the add-ins. The licensed add-ins will require the user to add a license file in the Installation Manager to be able to install the add-in.
Installation procedure

Before installing the software make sure that the certificates are available, for more information see Digital signing on page 80.

1. Install the RobotWare Add-In Packaging tool.
2. Install the certificate for signing add-ins using the RobotWare Add-In Packaging tool. Use the password provided by ABB.
   (A certificate is only needed when packaging licensed add-ins.)
3. Install your own publisher certificate.
   (A certificate is only needed when packaging licensed add-ins.)
4. Start the RobotWare Add-In Packaging tool.
2.5.1.2 Optional features

Option identity

The option identity is what uniquely identifies an option in a product.

The option identity namespace must start with the product identity and also have its own unique part. If the add-in has many options, the option identity part may be built up of several parts, to group options logically.

For example: open.yourcompany.yourproduct.youroption

When you decide what scheme to use for the option identity names, keep in mind that these option identity names are the identifiers that will be used in settings files and license files (for licensed add-ins). If option identifiers are changed between two releases of an add-in, compatibility with old settings files and license files will be broken.

System options and robot options

In RobotWare 7 there is support for both system options and robot options. Typically an option is classified as a robot option if its primary use is within the task of a robot. For example, equipment that a robot is dressed with is an example of a robot option. Or something that is connected to, or set up for, a specific robot in a MultiMove system. A system option is global to the system, for example languages.

Dependencies

A dependency specified for an option in an add-in could be either of type AND dependency, or of type OR dependency. This will define the dependency rule between the options selected.

For example, dependencies like the following can be defined: Source option A is dependent on both option B and C. Source option D is dependent on either A, B, or C.

AND dependency

If an option does not work unless all of its dependent options are also being installed, all these options are mandatory and should go into the AND dependency list.

Example:

813-1 Optical Tracking
<AND dependent on>
   624-1 Continuous Application Platform
   628-1 Sensor Interface

OR dependency

If an option does not work unless one of its dependent options are also being installed, all these options should go into the OR dependency list. In this case the option will work if either of the options in the list are also selected for installation.

Continues on next page
For example, PROFlenergy requires that either PROFINET Controller/Device or PROFINET Device is selected for installation:

963-1 PROFlenergy
<OR dependent on>
888-2 PROFINET Controller/Device
888-3 PROFINET Device
2.5.1.3 Files of a packaged add-in

The product manifest file

The product manifest file (.rmf) is a container of the metadata for the add-in product. It contains all product and option details.

Product details:
- Product name, product id, product version, version name, company name, company url, copyright, and description.
- Any product dependencies to other products, such as RobotWare or add-ins that the product may have.

Option details:
- Descriptions of all the options that are included in the add-in, such as option names, option id’s, option type (system or robot) and licensing restrictions,
- How the option structure should be displayed to the user in the Installation Manager.
- Any dependencies to other options that the options in the add-in may have.
- Any conflicts to other options that the options in the add-in may have.

The purpose of the product manifest is to define how the end-user will see the product in the Installation Manager. It will display the options in a structure to the user and define the rules for how options can be selected and what other products are required for the add-in to work.

The robot package file

The robot package file (.rpk) is an archive file that contains the actual contents of the add-in, in a compressed form.

The folders and files of the add-in containing installation and application logics in .cmd, .cfg, and .sys files.

This package will be transferred to the controller during installation and will be unpacked on the controller where the .cmd files of the add-in will be executed to install the add-in on the controller.
2.5.1.4 Signing with digital certificates

Digital signing

RobotWare 7 uses signing with digital certificates to ensure the integrity of published products. When creating a RobotWare add-in that contains licensed options a digital signature is mandatory.

To digitally sign a RobotWare add-in two different types of certificates are required, a publisher certificate and a licensing certificate.

The publisher certificate signature has 2 main purposes:

- Identify the publisher of the add-in to the end user.
- Ensure the integrity of the published software. For example, any modifications to the signed product manifest file will make the signature invalid and cause the robot controller to refuse to install the add-in.

The publisher certificate is also commonly known as a code-signing certificate. The add-in packaging tool will accept any x509 v3 certificate issued for this purpose.

ABB does not issue publisher certificates, it is the responsibility of the add-in developer to obtain a suitable certificate for example by purchasing it from a trusted certificate authority vendor or create their own self-signed certificate.

The licensing certificate is issued by ABB. This certificate is tied to the product id you specify and grants you as the publisher the right to issue licenses for your add-in. In addition to being used to sign your product the licensing certificate is also used by the License Generator when creating license files for your RobotWare add-in.

Timestamping

In addition to the signing certificates the RobotWare Add-In Packaging tool also allows you to specify a timestamping server. Timestamping is the process of applying a timestamp from a trusted source to your digital signature. This ensures that the signature will still remain valid even if the signing certificate expires or is revoked at a later date.

For example, without a timestamp the act of revoking a publisher or licensing certificate would invalidate all products ever signed with these certificates whereas with a timestamp products signed up to the revocation date will still remain valid.

Although not required, it is considered best practice and recommended to apply a timestamp when signing your product.

The RobotWare Add-In Packaging tool supports timestamping services that follows Microsoft Authenticode® standard. If you have purchased a publisher certificate from a certificate authority they should be able to recommend a suitable timestamping service.

As an alternative Symantec® operates a public timestamping service at the URL http://timestamp.verisign.com/scripts/timestamp.dll. (Note that it is not possible to browse to this URL.)
Installation of digital certificates

All digital certificates (with the exception of self-signed certificates) are signed by an issuer certificate. The issuer certificate in turn can have its own issuer, and so on, until a self-signed root certificate is reached, this forms a so called certificate chain.

For example the certificate chain for an ABB issued licensing certificate looks like this:

```
ABB RobotWare Licensing Root
 |  
ABB RobotWare Licensing Issuing CA
 |  
Licensing for <your product>
```

The add-in packaging tool requires that all issuer certificates must be installed in the Windows certificate store to be able to use the end user certificate for signing. In the example above the `ABB RobotWare Licensing Root` and `ABB RobotWare Licensing Issuing CA` certificates must be installed in order to be able to use the `Licensing for <your product>` certificate.

In the case of publisher certificates, if you have purchased a certificate from a 3rd party vendor the necessary certificate chain is usually already preinstalled in Windows and no further installation is necessary.

In the case of licensing certificates the complete certificate chain is included in the `.pfx` file delivered from ABB and the simplest way to install the issuer certificates is therefore to install the `.pfx` file. This will also install the end user certificate which can be uninstalled afterwards if desired.

To install the certificates locate the `.pfx` file in Windows Explorer, right click on the file and select the `Install PFX` option, this will open up the Certificate Import Wizard.
Proceed through the wizard (you will need the pfx password provided by ABB) until prompted to select a certificate store:

By default the wizard will try to determine the appropriate store based on the type of certificate. This would cause parts of the certificate chain to be installed as a trusted root certificate which is not recommended in the case of licensing certificates for security reasons. Instead it is recommended to change the default option and place all the certificates in the personal store. This will not affect the signing operations but will prevent the certificates from being trusted for operations for which they are not intended.
Viewing the installed certificates

It is possible to view and manipulate the contents of the Windows certificate store through the `certmgr` snap-in to the Microsoft Management Console. To launch the snap-in, execute the file `certmgr.msc` in the Windows system folder, usually `C:\Windows\system32\certmgr.msc.`
2.5.1.5 Types of add-in packaging tools

Overview

The RobotWare Add-In Packaging tool is available in two forms; a GUI based tool and a console based packaging tool. See User interface on page 85 and Building an add-in from the console on page 101.
2.5.2 User interface

2.5.2.1 The home page

The home page of RobotWare Add-In Packaging tool is displayed when you select New or Open from the File menu.

The home page has three main views, Product Manifest, Files and Folders, and Signing Certificates.

When all the information about the add-in has been entered, the add-in is built by selecting Build Project from the Build menu.
### 2.5.2.2 The File menu

The File menu is used to manage the projects:

![Image of the File menu](image-url)

The following table provides an overview of the options available in the File menu:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Creates a new add-in project. The following options are available:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Empty 6.x project</strong>: Select this option to create a 6.x add-in package project from scratch.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Empty 7.x project</strong>: Select this option to create a 7.x add-in package project from scratch.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Project from an existing folder hierarchy</strong>: Select this option to create an add-in using an existing folder hierarchy.</td>
</tr>
<tr>
<td></td>
<td>For details about creating a project, see <a href="#">Creating and building an add-in project on page 100</a>.</td>
</tr>
<tr>
<td>Open</td>
<td>Opens an existing add-in project.</td>
</tr>
<tr>
<td></td>
<td><img src="note-url" alt="Note" /> The add-in project file extension is .rpkproj</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the current active project.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves the current active project.</td>
</tr>
</tbody>
</table>
**2 Reference material**

### 2.5.2.2 The File menu

*Continued*

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save As</td>
<td>Save the current active project to a different location on the file system/network.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>The <strong>Save As</strong> operation saves all the project details in project files (.rpkproj, .rpkspecs and .manifest) into the newly selected location. Also a copy of source files under the Files and folders tab will be created and stored under the newly selected project folder.</td>
</tr>
<tr>
<td>Recent Projects</td>
<td>Displays a list of 10 recently closed projects. You can choose to open a recent project directly, instead of using the Open menu item.</td>
</tr>
<tr>
<td>Exit</td>
<td>Exits the tool.</td>
</tr>
</tbody>
</table>
2.5.2.3 The Product Manifest view

Introduction

The Product Manifest view is used to fill the product related information that goes into the product manifest file. For example, product details such as Product Name, Company Name and Product Version. The Product Manifest view is also used to structure the add-in, and to set any dependencies or conflicts with the other add-ins or RobotWare versions.

![Product Manifest View]

Product Details tab

The following information is to be defined in the Product Details tab:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Name</td>
<td>The name of the product.</td>
</tr>
<tr>
<td>Product Identity</td>
<td>The internal identifier of the product.</td>
</tr>
<tr>
<td></td>
<td>The product id is what uniquely identifies the product.</td>
</tr>
<tr>
<td></td>
<td>For licensed products the Product Id must start with one of the namespace</td>
</tr>
<tr>
<td></td>
<td>strings defined by the licensing certificate. For more information, see</td>
</tr>
<tr>
<td></td>
<td>Digital signing on page 80.</td>
</tr>
<tr>
<td></td>
<td>For unlicensed products the Product Id must start with the string open,</td>
</tr>
<tr>
<td></td>
<td>for example: open, yourcompany, yourproduct.</td>
</tr>
</tbody>
</table>
### Field name | Description
---|---
**Product Version** | The product version field is used to uniquely identify a specific build of the product. This information is used by the Installation Manager and other tools to determine the relation between different releases of a product, that is, older, equal, or newer. The format of the product version follows the standards for Semantic Versioning 2.0.0:


**Note**
The format differs between RobotWare 6 and 7.

**Version Name** | The version name field represents the product version as displayed to the end user. It differs from the Product Version field in that it is intended for display purposes only and is not restricted to a specific format. It can, for example, contain identifiers such as "Beta" or "Release Candidate" in addition to the version.

For example, if the Product Version is "2.1.0-32.Internal.Beta1+32" the Version Name can be "2.1.0-32.Internal.Beta1".

**Note**
Add-Ins built with version 1.3 or older of the RobotWare Add-In Packaging tool are displayed in the Installation Manager as a combination of the Product Name and Product Version fields. From version 1.4 the Version Name is used instead of Product Version and it is therefore important that this field contains relevant information.

**Company Name** | The name of your company.
**Company Url** | The website of your company.
**Copyright** | Copyright information.
**Description** | Short product description.

The **Product Dependencies** settings are used to set up dependencies to other add-ins and RobotWare versions.

Click **Add** and then **Import** to add a dependent software. The following fields will be filled automatically:

### Field name | Description
---|---
**Identity** | The internal identifier of the product.
**Name** | The name of the product.
**Platform** | The installation target platform, for instance robot controller and/or virtual controller.
**Publisher** | The company name of the add-in publisher.
**MinVersion** | The minimum product version.
**MaxVersion** | The maximum product version (optional).
**Type** | Product type. Always set to **Add-In**.
The Options tab helps you to create options and to specify their details. Click New to display the required fields for creating a new option.

The information is to be defined in the following fields:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Displays all fields that must be completed for the creation of a new option.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the selected option.</td>
</tr>
<tr>
<td>Validate</td>
<td>Validates the newly created option.</td>
</tr>
<tr>
<td>Display Name</td>
<td>Type the name of the option. This name is displayed in the Installation Manager in RobotStudio.</td>
</tr>
<tr>
<td>Identity</td>
<td>Type the internal identifier of the option. This id is what uniquely identifies an option in a product. The identifier must begin with the internal identifier of the product. For example: open.yourcompany.yourproduct.youroption. For more information, see Optional features on page 77.</td>
</tr>
</tbody>
</table>
| Type       | Select the type of the option:  
- System - Options that are global for the system.  
- Robot - Options that can be set per robot in the system. For example, when using MultiMove where different robots may have different equipment. |
### Attributes
Select the option attributes:
- **Required license** - The option requires a license.
- **Is internal** - The option is not shown in the Installation Manager GUI.
- **Is default selected** - The option is selected by default in the Installation Manager in RobotStudio.
- **Is locked** - The option cannot be deselected in Installation Manager in RobotStudio.

**Note**
For licensed products, at least one option should have the **Required license** check box selected.

### Min Instances
The minimum number of robots in the system that can have the option.

**Note**
This field is only valid for the option type **Robot**.

### Max Instances
The maximum number of robots in the system that can have the option. For example, when using several robots in a MultiMove system.

**Note**
This field is only valid for the option type **Robot**.

---

### Validate the option

Before leaving the **Options** tab, you must validate the options by clicking the **Validate** button.

The following validation is performed:
- The option identity must always begin with product identity text as prefix.

### Feature Data

For each option it is possible to define key values that can be retrieved from `install.cmd` file during product installation. For more information see, *getkey on page 53*.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Id of the key value.</td>
</tr>
<tr>
<td>Key</td>
<td>Key value.</td>
</tr>
</tbody>
</table>
Feature Data in MultiMove systems

By default, all the robots in a MultiMove system will get the same option settings. When it is desired to have different settings for the different robots it is necessary to provide more details to the robot options in the Feature Data settings.

Select a robot option in the option view, in the Feature Data section, add {{instance}} to the Id or Key data of those robot options you would like to work per robot in a MultiMove system, for example ROBOT{{instance}}COLOR.

During installation, the Installation Manager will resolve {{instance}} to 1, 2, 3, or 4, depending on which robot this setting was meant for. This will allow to check for settings like ROBOT1COLOR, ROBOT2COLOR, ROBOT3COLOR, and ROBOT4COLOR in the install.cmd files, for example in the following way:

```
getkey -id "ROBOT1COLOR" -var 10 -strvar $ANSWER -errlabel ERROR
goto -strvar $ANSWER
#ORANGE
config ...
#NEXT
#ERROR
```
Categories tab

The Categories tab is used to group and structure the options according to how the add-in should be displayed in the Installation Manager. It is not allowed to mix system options and robot options within the same category. When both system options and robot options are included in the add-in, they must be put into separate categories.

<table>
<thead>
<tr>
<th>Categories tab</th>
<th>Installation Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="categories_tab.png" alt="Image" /></td>
<td><img src="installation_manager.png" alt="Image" /></td>
</tr>
</tbody>
</table>

The following validation is performed:

- Only the same option type can be grouped together in a category. That is, an option of the type System cannot be in the same category as an option of type Robot.

Dependency tab

The Dependency tab is used to configure the dependencies between options.

<table>
<thead>
<tr>
<th>Dependency tab</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="dependency_tab.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Continues on next page
A dependency specified for an option in an add-in could be either of type **AND dependency options**, or of type **OR dependency options**.

For more information, see *Dependencies on page 77.*

**Note**

Combining **AND dependencies** with **OR dependencies** in the same group is not supported.

Use the following procedure to configure the dependencies between options:

1. Select a source from the **Source** list. The source option is the option that should have a dependency to one or several other options.

2. Select an option in the list, and click **Add** to move the dependencies for that option either to the **AND dependency options** list or to the **OR dependency options** list.

**Note**

If you added a product dependency in the **Product Details** tab, the options of that product will also be listed as options that the source option can depend upon.

3. Click **Add Group** to define the dependency.

The group is added to the **Preview dependency groups** section.

**Note**

When the dependency has been defined, it is listed in the dependency group list. Use the **Edit Group** and **Remove Group** buttons to edit or remove a dependency rule for an option dependency group.
**Conflict tab**

The Conflict tab is used to configure conflicts between the options.

![Conflict tab image](image)

By configuring the conflicts, the conflicting options cannot be selected at the same time in the Installation Manager.

Add the conflicting options one by one, and group them by clicking **Add Group**. Create a conflict group for each set of conflicting options.

**Note**

Sometimes, options specified in an *OR dependency* list are also in conflict with each other. In that case they should also be added both to the *OR dependency* list and to a conflict group.

**The Files and folders view**

The Files and Folders view is used to create the robot package file.

**Note**

Verify that all the files and folders to be transferred to the controller during installation are in place. Files and folders can be added and removed using the user interface.

Files and folders can be added to the project using the Files and Folders view.
Right-click at the folder level for the following options:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Folder</td>
<td>Creates a new folder. The folder is added to the respective level on the tree.</td>
</tr>
<tr>
<td>Add Folder</td>
<td>Adds an existing folder on the file system to the project.</td>
</tr>
<tr>
<td>Add Files</td>
<td>Adds the individual files to the project.</td>
</tr>
<tr>
<td>Rename</td>
<td>Renames the selected folder.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>The root folder cannot be renamed.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the selected folder.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>The root folder cannot be removed.</td>
</tr>
</tbody>
</table>
Right-click inside a folder for the following options:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Folder</td>
<td>Creates a new folder under the selected folder. The folder is added to the respective level on the tree.</td>
</tr>
<tr>
<td>Add Folder</td>
<td>Adds an existing folder under the selected folder.</td>
</tr>
<tr>
<td>Add Files</td>
<td>Adds the individual files to the project.</td>
</tr>
<tr>
<td>Copy Full Path</td>
<td>Copies the full path of the selected file to the clipboard.</td>
</tr>
<tr>
<td>Open Containing Folder</td>
<td>Opens the selected folder location in Explorer.</td>
</tr>
<tr>
<td>Open</td>
<td>Opens the selected file in the software tool for the file.</td>
</tr>
<tr>
<td>Rename</td>
<td>Renames the selected file.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the selected file from the project.</td>
</tr>
</tbody>
</table>

The name of the installation folder is a combination of the **Product Name** and the **Product Version**, that was defined in the **Product Details** tab.

**Note**

The added files or folders are not physically copied to the project folder. The RobotWare Add-In Packaging tool creates only a reference to the source files or folders. Hence the added files and folders should be available at the original source path.

When the project files or folders are modified in the original source location, there will be impacts in the **Files and Folders** view while opening a saved project.

- if a file or folder is deleted from the source location, then there will be an indication about the missing file or folder in the **Files and Folders** view.
• if a file or folder is manually added to the source location, then no indication is provided. You need to manually add the new file or folder in the Files and Folders view of the RobotWare Add-In Packaging tool, if the newly added file or folder is needed in the output package.

Files and folders for converted add-ins

After converting an additional option to an add-in, the Syskey directory can be removed from the Files and Folders view, since it will no longer be used in the RobotWare 6 installation. It was required for the import of the additional option, since it enables the RobotWare Add-In Packaging tool to auto generate the option details, but now the folder can be removed.

The relkey.txt file can also be removed, since it is not used anymore.

The Signing Certificate view

The Signing Certificate view is used to add the publisher and licensing certificates. This information is mandatory for licensed options and is used during the signing of the manifest and robot package files. For more information, see Digital signing on page 80.

xx2000002005
<table>
<thead>
<tr>
<th>Section</th>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publisher Certificate</td>
<td>Select from file</td>
<td>Select this option if the publisher certificate for digital signing should be provided as .pfx files. <strong>Note</strong> Browse Publisher Certificate to select the certificate from its stored location.</td>
</tr>
<tr>
<td></td>
<td>Select from Windows certificate store</td>
<td>Select this option if the publisher certificate for digital signing should be installed on your PC from the Windows certificate store.</td>
</tr>
<tr>
<td></td>
<td>Publisher Certificate</td>
<td>Browse to select a certificate (.pfx file) from its stored location. The selected path is displayed. <strong>Note</strong> This field is used in combination with option Select from file.</td>
</tr>
<tr>
<td></td>
<td>Publisher Certificate</td>
<td>The password for the publisher certificate when specified as a .pfx file.</td>
</tr>
<tr>
<td>Licensing Certificate</td>
<td>Select from file</td>
<td>Select this option if the licensing certificate for digital signing should be provided as .pfx files. <strong>Note</strong> Browse Licensing Certificate to select the certificate from its stored location.</td>
</tr>
<tr>
<td></td>
<td>Select from Windows certificate store</td>
<td>Select this option if the licensing certificate for digital signing should be installed on your PC from the Windows certificate store.</td>
</tr>
<tr>
<td></td>
<td>Licensing Certificate</td>
<td>Browse to select a certificate (.pfx file) from its stored location. The selected path is displayed. <strong>Note</strong> This field is used in combination with option Select from file.</td>
</tr>
<tr>
<td></td>
<td>Licensing Certificate</td>
<td>The password for the licensing certificate when specified as a .pfx file.</td>
</tr>
<tr>
<td></td>
<td>Timestamp Server</td>
<td>Displays the URL to a timestamp server. For more information, see Timestamping on page 80.</td>
</tr>
</tbody>
</table>
2.5.3 Creating and building an add-in project

Procedure

Use the following procedure to create and package the add-in.

1. Create a new empty project by clicking **New** in the **File** menu, and then selecting one of the following options:
   - **Empty 6.x project**: Select this option to create a 6.x add-in package project from scratch.
   - **Empty 7.x project**: Select this option to create a 7.x add-in package project from scratch.
   - **Project from an existing folder hierarchy**: Select this option to create an add-in using an existing folder hierarchy. The tool will try to generate default data for the add-in.

2. Complete all mandatory information for the add-in in the **Product Manifest** view. This includes information regarding product details, options, categories, dependencies and conflicts. See *The Product Manifest view on page 88* for details.

3. Create the robot package file by adding files and folders in the **Files and Folders** view. See *The Files and folders view on page 95* for more details.

   **Note**

   Verify that all the files and folders to be transferred to the controller during installation are in place. Files and folders can be added and removed using the user interface.

4. For licensed options, add the publisher and licensing certificates in the **Signing Certificate** view. See *The Signing Certificate view on page 98* for more details.

5. Build the add-in by selecting **Build Project** from the **Build** menu.

6. Generate a license using the License Generator. See *License Generator on page 103* for more details.

7. Verify the add-in by building a system using the Installation Manager in RobotStudio. See *Operating manual - RobotStudio* for more information.
2.5.4 Building an add-in from the console

Overview

The console version of the RobotWare Add-In Packaging tool, APTCommandLine.exe, is used to build an existing add-in project from the command line.

The console version may be used as a batch command with relevant information to generate the add-in.

Use the argument "-h" along with APTCommandLine.exe to display all the available arguments.

Note

Use : (colon) to separate an argument name and its value.

Note

Run APTCommandLine.exe without any argument on the command line to view the usage of arguments with examples.

Prerequisites

The add-in project must be created with all relevant references and desired files and folders using the with the GUI based add-in packaging tool.

The console based add-in packaging tool uses this project to generate the add-in in an unattended manner when provided with all the relevant information in the batch command.

Description

The following table provides details of allowed add-in packaging tool command line parameters switches:

<table>
<thead>
<tr>
<th>Parameters switches</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-projectFileName</td>
<td>Project file name for APT RPKProj file.</td>
</tr>
<tr>
<td>-pubcertfile</td>
<td>Publisher signing certificate file.</td>
</tr>
<tr>
<td>-pubcertfilepassword</td>
<td>Password for the publisher certificate.</td>
</tr>
<tr>
<td>-liccertfile</td>
<td>Licensing signing certificate file.</td>
</tr>
<tr>
<td>-liccertfilepassword</td>
<td>Password for the licensing certificate</td>
</tr>
<tr>
<td>-liccertthumbprint</td>
<td>Thumbprint for the licensing certificate stored in the certificate store.</td>
</tr>
<tr>
<td>-pubcertthumbprint</td>
<td>Thumbprint for the publisher certificate stored in the certificate store.</td>
</tr>
<tr>
<td>-timestampurl</td>
<td>Timestamping server URL for code signing.</td>
</tr>
<tr>
<td>-outputfolder</td>
<td>Output folder where project output will be generated.</td>
</tr>
<tr>
<td>-isopenaddin</td>
<td>If this parameter's value is set to TRUE, an open add-in is generated without considering the licensing.</td>
</tr>
</tbody>
</table>

Continues on next page
For signing APT output files using Certificate files, possible options are:
- Publisher certificate files `-pubcertfile` along with the certificate file password `-pubcertfilepassword`.
- Licensing certificate files `-liccertfile` along with the certificate file password `-liccertfilepassword`.

For signing APT output files with thumbprint of certificate in the computer’s certificate store, possible options are:
- Publisher thumbprint `-pubcertthumbprint`
- Licensing thumbprint `-liccertthumbprint`

**Note**

For publisher/licensing certificate signing, user can either use certificate file(s) and password or thumbprint(s) but not both in a single batch instruction.

**Note**

It is possible to use file certificate file and password for publisher signing and thumbprint for license signing.
2.6 License Generator

2.6.1 Introduction

**General**

The License Generator generates license files for RobotWare add-ins.

**Installation procedure**

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>The License Generator, including the license certificate, must be ordered from ABB.</td>
</tr>
</tbody>
</table>

1. Install the License Generator.
2. Install the certificate for the License Generator. Use the password provided by ABB.
3. Start the License Generator.
2 Reference material

2.6.2.1 The Preferences window

2.6.2 The user interface

2.6.2.1 The Preferences window

Preferences

Before running the License Generator, the preferences in the Preferences window must be set up:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Locations</td>
<td>The location of the product manifest files (*.rmf).</td>
</tr>
<tr>
<td>Default License Location</td>
<td>The default location where the created licence files (*.rlf) should be saved.</td>
</tr>
<tr>
<td>Authorization File</td>
<td>The authorization file, license file, (*.rlf) for the License Generator provided by ABB.</td>
</tr>
<tr>
<td>Signing Certificate</td>
<td>Install/use the certificate provided by ABB, same certificate as for the RobotWare Add-In Packaging tool.</td>
</tr>
<tr>
<td>Certificate Password</td>
<td>Use the certificate password provided by ABB.</td>
</tr>
<tr>
<td>Timestamp Server</td>
<td>URL to a timestamp server. For more information, see Timestamping on page 80.</td>
</tr>
</tbody>
</table>
2.6.2.2 The main window

Overview of the main window

The main window is used to add all options that are to be included in the license file. When all options are added, the license file is built by clicking Generate License.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select product</td>
<td>Select the product manifest for which the license should be created.</td>
</tr>
<tr>
<td>License</td>
<td>Select a license to import. The content of that license will be copied.</td>
</tr>
<tr>
<td>Serial Number</td>
<td>Enter the serial number of the controller.</td>
</tr>
<tr>
<td>Expand/collapse button.</td>
<td>Expand/collapse the options in the selected tab.</td>
</tr>
<tr>
<td>License Type</td>
<td>License type Controller is selected.</td>
</tr>
<tr>
<td>Clear All Selections</td>
<td>Clear all selected options.</td>
</tr>
<tr>
<td>Auto resolve dependencies</td>
<td>Automatically select dependant options.</td>
</tr>
<tr>
<td>Generate License</td>
<td>Generate the license file.</td>
</tr>
</tbody>
</table>

Tip

Double-click an option in the Preview Summary window to locate and highlight the option in the tree-view.
Tip

Use the search function to search for option names instead of browsing through the tree-view.
2.6.3 Creating the license

Creating a new license

Note

Before creating a license it is necessary to set up the preferences in the Preferences window, see The Preferences window on page 104.

Use this procedure to create a new license.

1. Start the License Generator.
2. Set up the preferences in the Preferences window, see The Preferences window on page 104.
3. In the main window, in field Select Product, select the product manifest for which the license should be created.
4. Enter the Serial Number of the controller.
5. In the tree-view, select all options to be included in the license.
6. Click Generate License to generate the license file.
7. Verify the license by building a system using the Installation Manager in RobotStudio.

Import and modify a license

Before creating a license it is necessary to set up the preferences in the Preferences window, see The Preferences window on page 104.

Use this procedure to import and modify a license.

1. Start the License Generator.
2. Set up the preferences in the Preferences window, see The Preferences window on page 104.
3. In the main window, in field Select Product, select the product manifest for which the license should be created.
4. In field License, select the license to be imported and then click Open.
5. Enter the Serial Number of the controller.
6. In the tree-view, add or remove options.
7. Click Generate License to generate the license file.
8. Verify the license by building a system using the Installation Manager in RobotStudio.

Viewing a licence file

The content of the license file is displayed in the Licence View window.

Use this procedure to view a license.

1. Start the License Generator.
2. In the main window, click View to access the Licence View window.
3. Browse to the folder where the license is located.
4. Select the license file and click Open.

Continues on next page
5 The content of the license file is displayed.
A Appendix: Migration from RobotWare 6

RobotWare 6 install script command migration to RobotWare 7

The following table displays the RobotWare 6 commands and their migration status to RobotWare 7:

- **Not changed:**
  RobotWare 7 command is compatible with RobotWare 6 command.

- **Available with restrictions:**
  The RobotWare 7 command has path restrictions compared to the RobotWare 6 command:
  An add-in install script can only read from and write to the paths specified in *Introduction on page 44*. The product installation directory is read-only and add-ins cannot remove the write protection. See environment variables under *Introduction on page 44* for more details.

- **Replaced:**
  The command is not available in RobotWare 7 but a replacement command with corresponding functionality is introduced in RobotWare 7.

- **Not available:**
  Currently not available in RobotWare 7. Use cases are requested for this command.

- **Removed:**
  Actively removed and there are no plans to support it in RobotWare 7.

<table>
<thead>
<tr>
<th>Name</th>
<th>Migration status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addvar</td>
<td>Replaced</td>
<td>Replaced by <em>addintvar on page 48</em>.</td>
</tr>
<tr>
<td>append</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See <em>Introduction on page 44</em>.</td>
</tr>
<tr>
<td>attrib</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See <em>Introduction on page 44</em>.</td>
</tr>
<tr>
<td>config</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>copy</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See <em>Introduction on page 44</em>.</td>
</tr>
<tr>
<td>delay</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See <em>Introduction on page 44</em>.</td>
</tr>
<tr>
<td>delete_cfg_instance</td>
<td>Not available</td>
<td>Currently not available in RW 7.1. Can be considered for RW 7.2. Use cases are requested for this command.</td>
</tr>
<tr>
<td>direxist</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See <em>Introduction on page 44</em>.</td>
</tr>
<tr>
<td>echo</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>fileexist</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See <em>Introduction on page 44</em>.</td>
</tr>
<tr>
<td>find_replace</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See <em>Introduction on page 44</em>.</td>
</tr>
</tbody>
</table>

*Continues on next page*
<table>
<thead>
<tr>
<th>Name</th>
<th>Migration status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getkey</td>
<td>Not changed</td>
<td>Do not use argument -varno. Consider using command <code>if_feature_present</code> on page 53.</td>
</tr>
<tr>
<td>goto</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>if</td>
<td>Replaced</td>
<td>Replaced by <code>ifintvar</code> on page 54.</td>
</tr>
<tr>
<td>ifstr</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>ifvc</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>include</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See Introduction on page 44.</td>
</tr>
<tr>
<td>invoke</td>
<td>Removed</td>
<td>Available in RW 6, but not documented and supported.</td>
</tr>
<tr>
<td>mkdir</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See Introduction on page 44.</td>
</tr>
<tr>
<td>loop</td>
<td>Replaced</td>
<td>Replaced by <code>loop_include</code> on page 55 and <code>loop_break</code> on page 54.</td>
</tr>
<tr>
<td>onerror</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>print</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>rapid_delete_palette</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>register</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>rename</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See Introduction on page 44.</td>
</tr>
<tr>
<td>setenv</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>setstr</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>setvar</td>
<td>Replaced</td>
<td>Replaced by <code>setintvar</code> on page 59.</td>
</tr>
<tr>
<td>text</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>timestamp</td>
<td>Not changed</td>
<td></td>
</tr>
<tr>
<td>uas_install_application_grants</td>
<td>Not available</td>
<td>Currently not available in RW 7.1. Can be considered for RW 7.2. Uses cases are requested for this command.</td>
</tr>
<tr>
<td>uas_install_groups</td>
<td>Not available</td>
<td>Currently not available in RW 7.1. Can be considered for RW 7.2. Uses cases are requested for this command.</td>
</tr>
<tr>
<td>xattrib</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See Introduction on page 44.</td>
</tr>
<tr>
<td>xcopy</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See Introduction on page 44.</td>
</tr>
<tr>
<td>xdelete</td>
<td>Available with restrictions</td>
<td>Path restrictions apply. See Introduction on page 44.</td>
</tr>
</tbody>
</table>
Note

RobotWare 7 sandboxes all add-ins in a more strict way than RobotWare 6. Therefore, only the commands described in this manual and with the restrictions described here can be used in RobotWare 7.
ABB AB
Robotics & Discrete Automation
S-721 68 VÄSTERÅS, Sweden
Telephone +46 (0) 21 344 400

ABB AS
Robotics & Discrete Automation
Nordlysvegen 7, N-4340 BRYNE, Norway
Box 265, N-4349 BRYNE, Norway
Telephone: +47 22 87 2000

ABB Engineering (Shanghai) Ltd.
Robotics & Discrete Automation
No. 4528 Kangxin Highway
PuDong New District
SHANGHAI 201319, China
Telephone: +86 21 6105 6666

ABB Inc.
Robotics & Discrete Automation
1250 Brown Road
Auburn Hills, MI 48326
USA
Telephone: +1 248 391 9000

abb.com/robotics

© Copyright 2020-2022 ABB. All rights reserved.
Specifications subject to change without notice.