



ROBOTICS

Application manual

RobotWare add-ins



Trace back information:
Workspace 23C version a25
Checked in 2023-10-10
Skribenta version 5.5.019

Application manual

RobotWare add-ins

RobotWare 7.12

Document ID: 3HAC070207-001

Revision: F

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 2020-2023 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 Register safety template	25
2.3 System parameters related to add-in development	30
2.3.1 About cfg files	30
2.3.2 Topic Controller	33
2.3.3 Topic I/O System	36
2.3.4 Topic Man-machine Communication	37
2.3.5 Example cfg files	47
2.4 The install.cmd file	49
2.4.1 Introduction	49
2.4.2 Commands	53
2.4.3 Examples of install.cmd files	67
2.5 RAPID	68
2.5.1 RAPID modules	68
2.5.2 Using text resources from files	70
2.5.3 Hiding RAPID content	71
2.5.4 Optional settings for RAPID arguments (RAPID meta data)	73
2.5.4.1 Hiding arguments in programs	74
2.5.4.2 Hiding optional argument when changing selected instruction	76
2.5.4.3 Argument filter	78
2.5.4.4 Argument value range	80
2.6 RobotWare Add-In Packaging tool	81
2.6.1 Introduction	81
2.6.1.1 About the RobotWare Add-In Packaging tool	81
2.6.1.2 Optional features	83
2.6.1.3 Files of a packaged add-in	85
2.6.1.4 Signing with digital certificates	86
2.6.1.5 Types of add-in packaging tools	90
2.6.2 User interface	91
2.6.2.1 The home page	91
2.6.2.2 The File menu	92
2.6.2.3 The Product Manifest view	94
2.6.3 Creating and building an add-in project	106
2.6.4 Building an add-in from the console	107
2.7 License Generator	109
2.7.1 Introduction	109
2.7.2 The user interface	110
2.7.2.1 The Preferences window	110
2.7.2.2 The main window	111
2.7.3 Creating the license	113
A Appendix: Migration from RobotWare 6	115

Table of contents

B Appendix: Product manifest files guidelines (RobotWare 7)	119
Index	123

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 115](#) 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

References

Reference	Document ID
Technical reference manual - System parameters	3HAC065041-001
Operating manual - RobotStudio	3HAC032104-001
Operating manual - Integrator's guide OmniCore	3HAC065037-001

Revisions

Revision	Description
A	Released with RobotWare 7.1.
B	Released with RobotWare 7.2. <ul style="list-style-type: none">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 53.New script <code>math_lib_set_mem_size</code> added in section Commands on page 53.
C	Released with RobotWare 7.5. <ul style="list-style-type: none">Section Argument Name Rules (MMC_REAL_PARAM) on page 39 updated with information about how to add a string in Rapid rules.
D	Released with RobotWare 7.7. <ul style="list-style-type: none">Information regarding the <code>config</code> command updated in section Commands on page 53.Updated the section Building an add-in from the console on page 107.
E	Released with RobotWare 7.10. <ul style="list-style-type: none">Updated the section Commands on page 53.Updated the section config on page 54.Information about Product Identity updated in The Product Manifest view on page 94.Information about Product Id updated in The Product Manifest view on page 94.
F	Released with RobotWare 7.12. <ul style="list-style-type: none">Added the section Register safety template on page 25.Added the parameter <code>robot</code> to the section if_feature_present on page 59.Added the parameter <code>envvalue</code> in setstr on page 65.Command for I/O project installation added in The install.cmd file on page 49.New appendix: Appendix: Product manifest files guidelines (RobotWare 7) on page 119. Link added to new appendix in RobotWare Add-In Packaging tool on page 81.

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.



WARNING

Program changes should always be validated and tested before entering production, to protect humans and property. Ensure it is possible to stop the robot with a protective stop device.

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*.

This page is intentionally left blank

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.

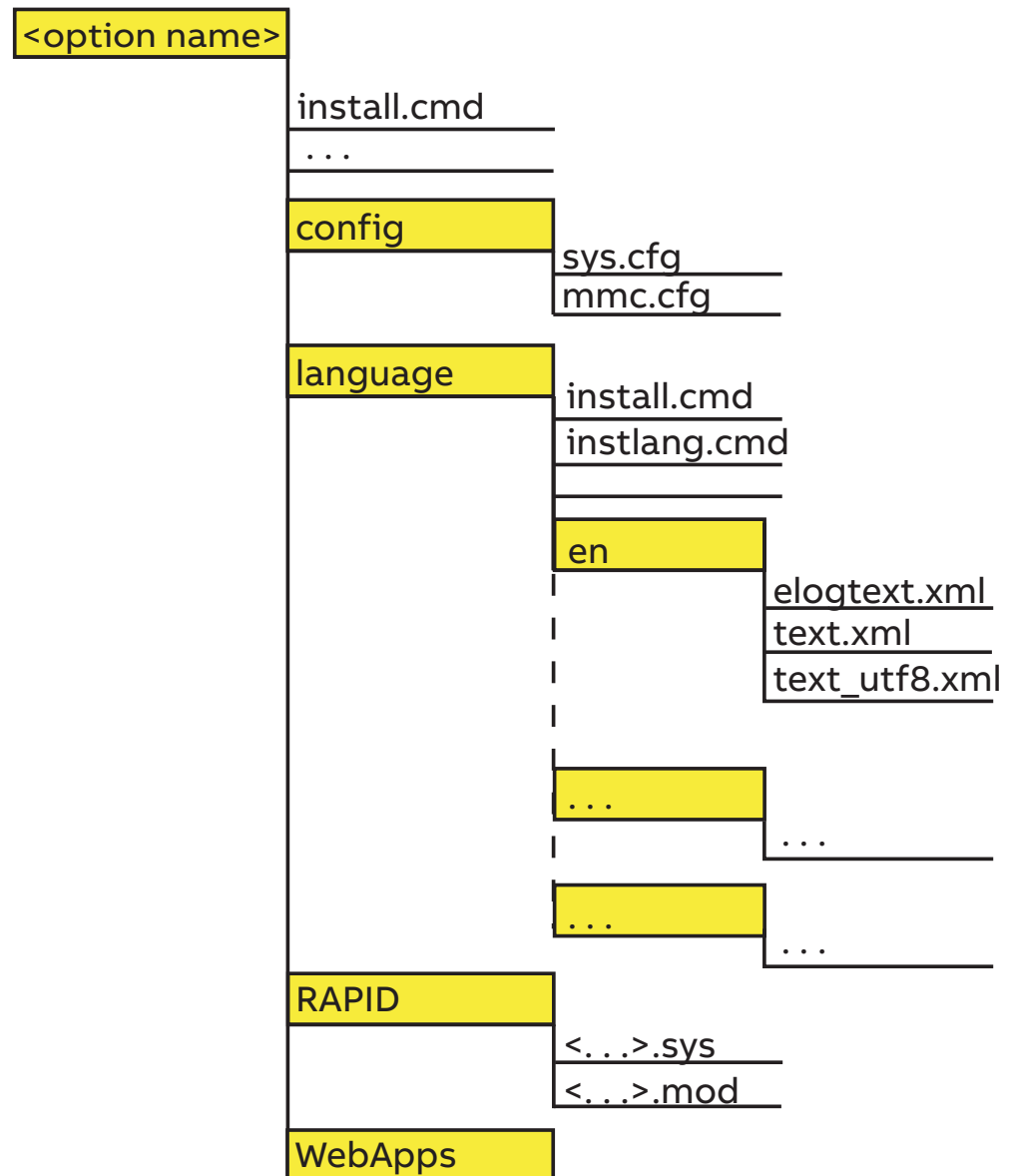
1 Getting started

1.2 Add-in directory and file structure

1.2 Add-in directory and file structure

Recommended file structure

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



xx2000002017

Add-in files

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

File type	Description
<i>install.cmd</i>	Installation script. Specifies for example which <i>.cfg</i> files to load, see The install.cmd file on page 49 . Must be in the root directory of your add-in.

Continues on next page

File type	Description
<i>config/</i>	One or several <i>.cfg</i> files with the configuration of system parameters. If the add-in includes RAPID, one of the <i>.cfg</i> files should specify which RAPID module (<i>.sys</i> file) to load, see System parameters related to add-in development on page 30 .
<i>language/</i>	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 70 .
<i>RAPID/</i>	The RAPID source code of your modules contained in files <i>.sys</i> or <i>.mod</i> , see RAPID modules on page 68 .
<i>WebApps</i>	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.

1 Getting started

1.3 Quick start procedures for example add-in

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:

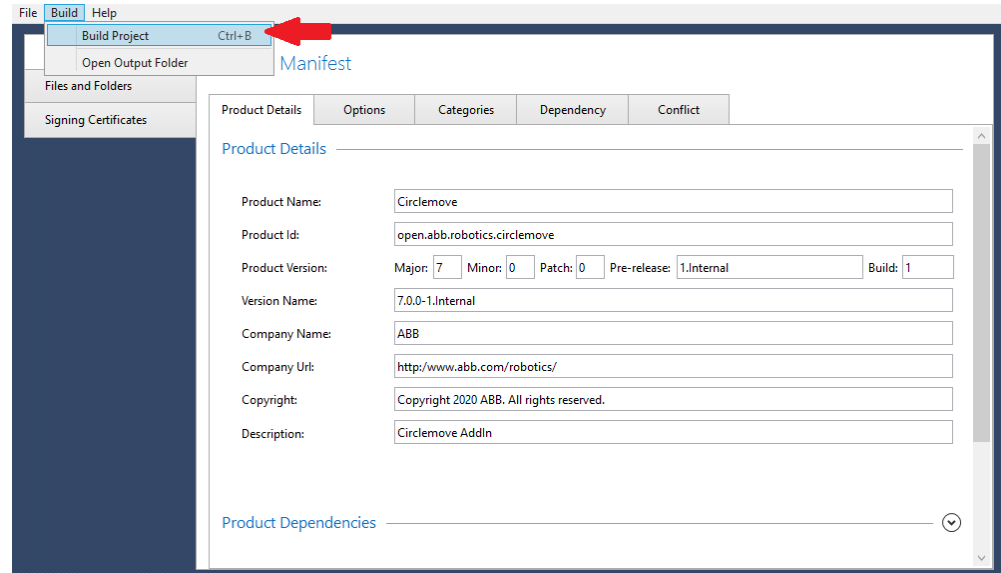
- Implements a RAPID command called *Circlemove*, which can be used by RAPID programs.
- Shows how to define and use custom event log messages.
- Shows how to use localizable text resources from your RAPID code.
- Implements a simple WebApp that shows up on the FlexPendant. No additional configuration 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 63](#).

See [Add-in directory and file structure on page 12](#) for location of the files.

Continues on next page

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**.



xx2000002045

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 the Modify Installation function

Modify Installation

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 the **Modify Installation** function.

Continues on next page

1 Getting started

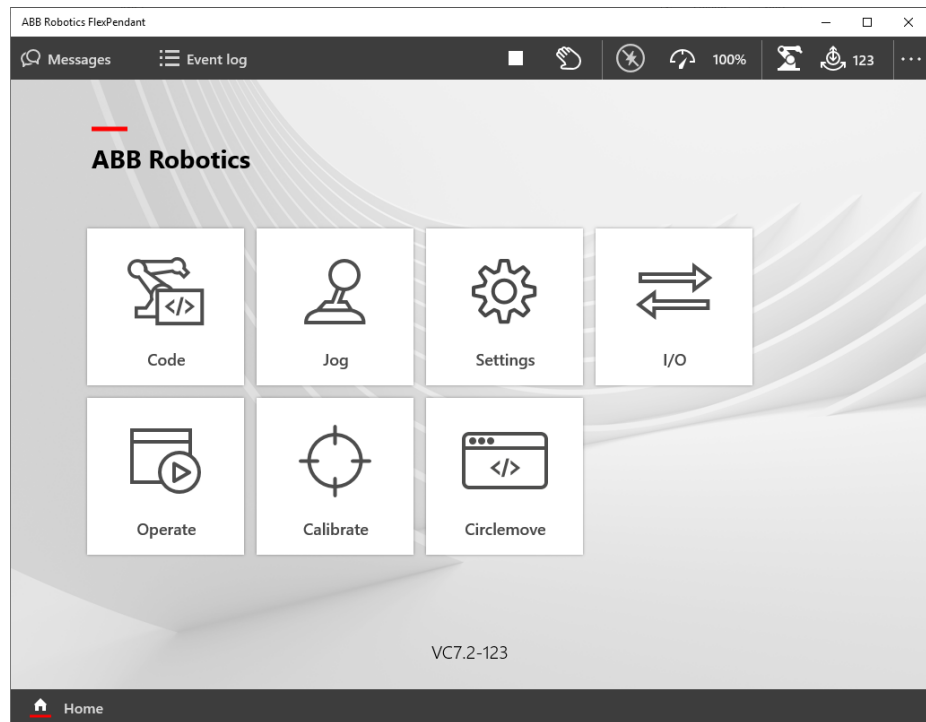
1.3 Quick start procedures for example add-in

Continued

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:

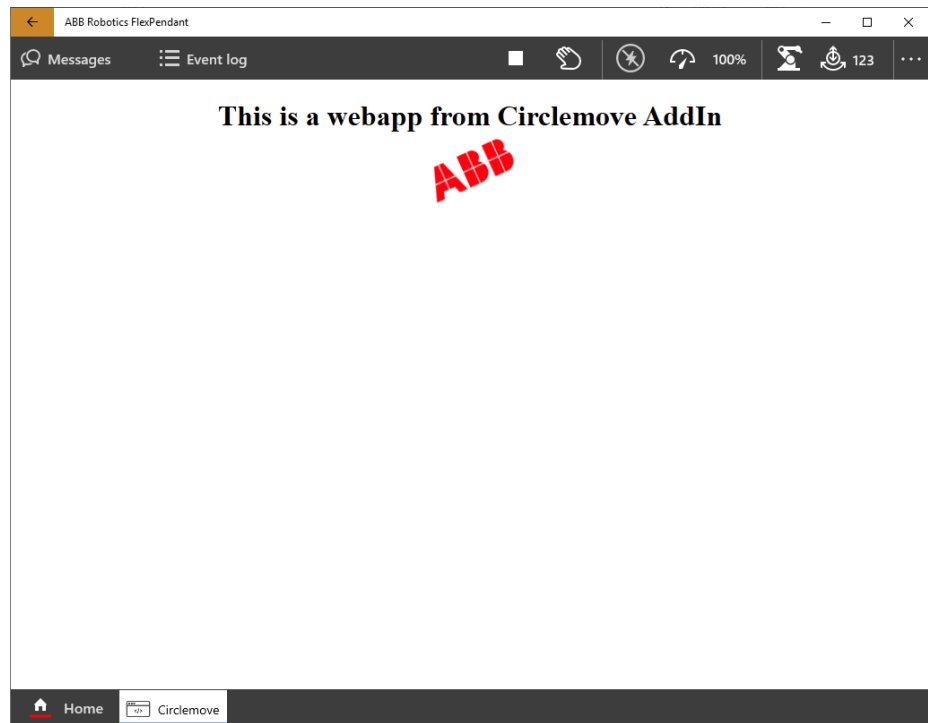
- 1 Select your Virtual Controller and start the Omnicore virtual FlexPendant that is connected to your virtual system. The **Circlemove** user interface shall appear on the start page of the FlexPendant:



xx2000002042

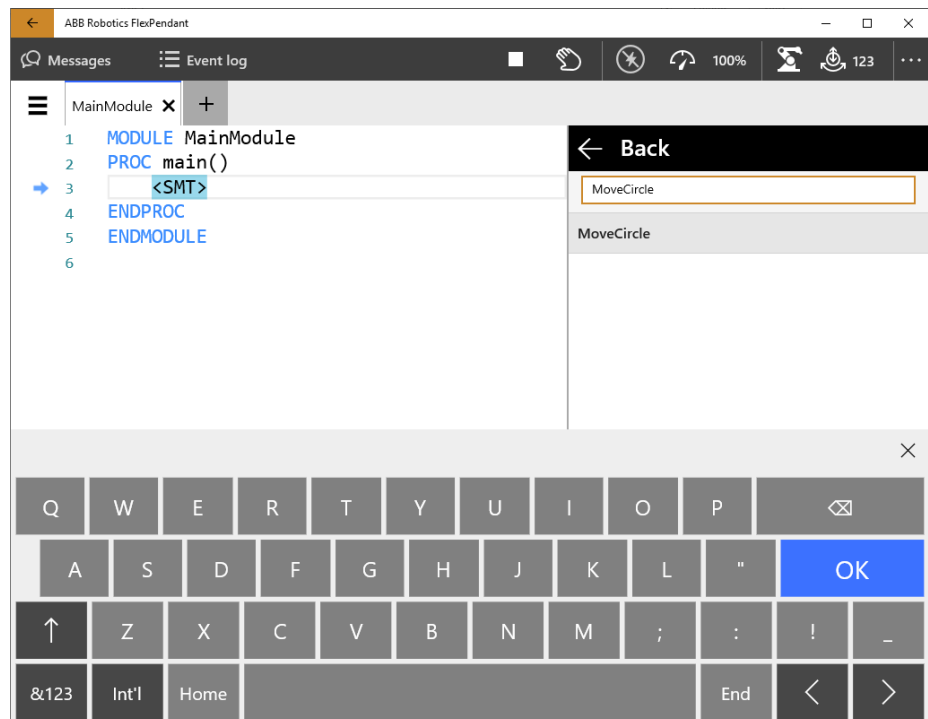
Continues on next page

- 2 Click on the **Circlemove** icon and the following page showing a rotating ABB logo shall appear:



xx2000002043

- 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:



xx2000002044

Continues on next page

1 Getting started

1.3 Quick start procedures for example add-in

Continued

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 **Modify Installation** function 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 **Modify Installation** function when creating a RobotWare system. To learn more on how to work with RobotStudio *rspack*, see the RobotStudio developer center documentation:

https://developercenter.robotstudio.com/api/robotstudio/articles/Concepts/Distribution-Package/DP_Overview.html

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 Reference material

2.1.2 Event log texts

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

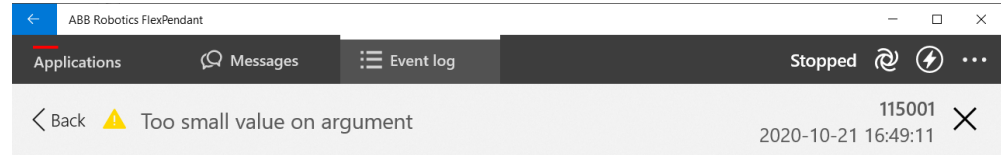
This is a list of the XML tags and arguments that you need to define. All other tags and arguments should always look like in the example below. The complete syntax is also shown in the example below.

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

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:

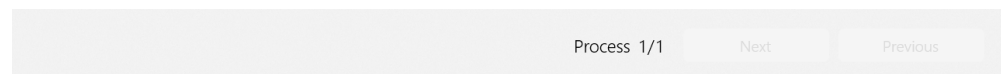


Details

Task: T_ROB1

The argument Radius was set to 1.00 but the minimum allowed value is 2.

Context: /MainModule/main/MoveCircle/4



xx2000002145

```
<?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: <arg format="%s" ordinal="1" /><p />
      The argument <arg format="%s" ordinal="2" /> was set to <arg
        format="%s" ordinal="3" /> but the minimum allowed value
        is
      <arg format="%s" ordinal="4" />. <p />
      Context: <arg format="%s" ordinal="5" />
    <p />
    </Description>
  </Message>
</Domain>
```

2 Reference material

2.1.3 Validating event log .xml files

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 [Robot-Studio 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

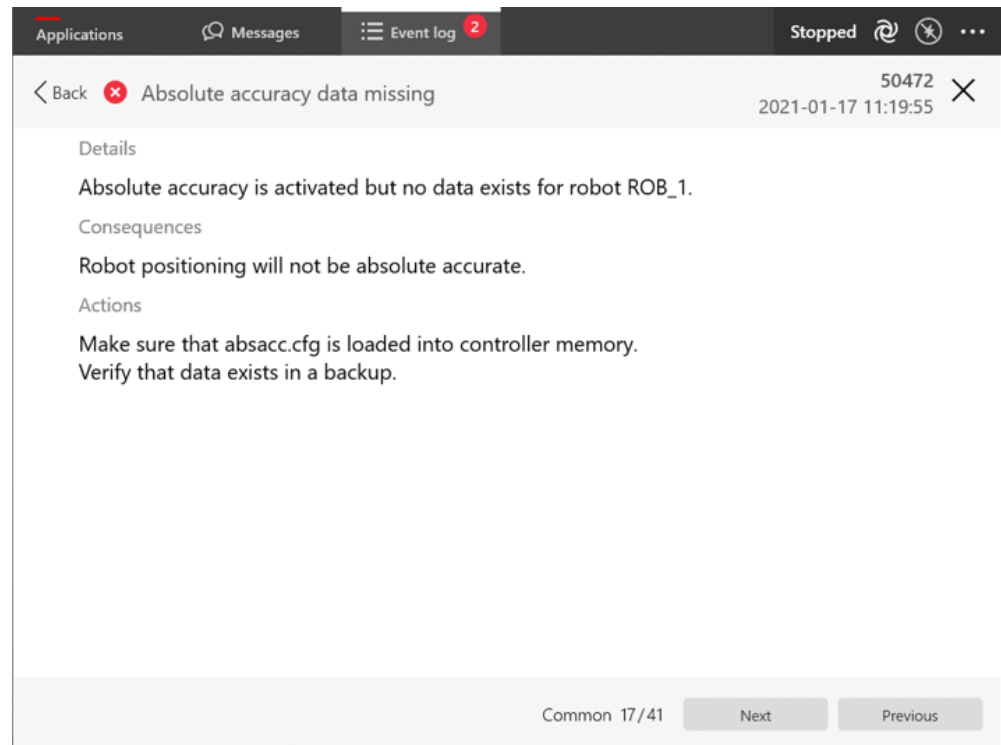
Overview

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:



xx2000002180

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:



xx2000002179

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](#).

Continues on next page

2 Reference material

2.1.4 Configure event logs to take focus on the FlexPendant

Continued



Note

This feature is only available for RobotWare 7.1 and onwards in combination with FlexPendant software version 1.3.x.

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 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 49](#).

Example

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

2.2 Register safety template

Introduction

Few safety configuration templates are included in FlexPendant when it is newly delivered. It is also possible to package a new safety configuration template.

The RobotWare add-in Packaging Tool helps you to create a package with safety configuration templates and this package can be distributed and installed into a RobotWare system. This feature allows you to package any number of safety configuration templates.

Add-in directory and file structure

An Add-in implementation consists of several files and directories. The following structure is recommended to install safety configuration templates into the RobotWare systems.

- Add-in root folder

- `install.cmd`

This is the installation script to register the safety configuration templates.

- Safety

- # `<....>_metadata.xml`

- The file contains information about the safety configuration templates

- # Templates

- The folder contains the number of safety configuration templates

- `<....>.xml`

- `<....>.xml`

Continues on next page

2 Reference material

2.2 Register safety template

Continued

The following directory and file structure is for reference. The structure can be organized in any way as long as it follows the add-in guidelines as described in [Add-in directory and file structure on page 12](#).

```
CRB1100_Templates
├── install.cmd
├── Safety
│   └── CRB1100_safety_templates_metadata.xml
└── Templates
    ├── CRB1100_047_IO2_SafeMove_Template.xml
    ├── CRB1100_047_IO_SafeMove_Template.xml
    ├── CRB1100_047_PROFI_DIRECT2_SafeMove_Template.xml
    ├── CRB1100_047_PROFI_DIRECT_SafeMove_Template.xml
    ├── CRB1100_047_PROFI_SafeMove_Template.xml
    ├── CRB1100_058_IO2_SafeMove_Template.xml
    ├── CRB1100_058_IO_SafeMove_Template.xml
    ├── CRB1100_058_PROFI_DIRECT2_SafeMove_Template.xml
    ├── CRB1100_058_PROFI_DIRECT_SafeMove_Template.xml
    └── CRB1100_058_PROFI_SafeMove_Template.xml
```

xx2300001520

Prepare the directory and files for the add-in

The following section describes suggested files and directories for the add-in.

`install.cmd`

`install.cmd` is a mandatory file, and it must be in the root directory of add-in. This script is executed when the system starts and registers a safety template.

Following is the script that registers safety templates in the RobotWare system.

```
register -type safety_template -prepath $BOOTPATH/ -postpath
Safety/<...>_metadata.xml
```

The script can be customized to work for a specific configuration, robot, and so on. Refer to the section [System parameters related to add-in development on page 30](#) for details about customizing the scripts.

Safety

Safety is not a mandatory directory, but it is good to create this directory to keep all the safety template related files and directories together.

For example, this directory can contain `<...>_metadata.xml` file and other directories needed for the safety template add-in.

Since Safety is not a mandatory directory, the files and directories can be placed directly in the root directory of add-in or in some other directory. But the path for the `<...>_metadata.xml` file must be correct in the script to register safety templates in `install.cmd`.

Continues on next page

<...>_metadata.xml

The <...>_metadata.xml file is mandatory and can be placed anywhere within the add-in directory but the path for the file must be correct in install.cmd.

This file contains a list of XML tags and arguments that you need to define. The following table gives an overview of all the XML tags or arguments.

XML tags or arguments	Description
xmlns	XML namespace for naming schema
Templates	It is a collection of Safety Configuration Templates
version	The version of the safety templates metadata format
Template	Create one instance of Template for each safety configuration template.
name	Unique friendly name for the Template instance
Title	The title for the Template instance will be shown in the SafeMove Template Selector page.
Description	The text describes the safety configuration template that will be shown in the SafeMove Template Selector page.
Date	When the safety configuration template is created that will be shown in the SafeMove Template Selector page.
TemplateFile	This helps to locate where the actual safety configuration template is stored.
path	<p>The path attribute in <TemplateFile/> shall be either relative to <...>_metadata.xml file or the complete path.</p> <p>Relative path: The path for the safety configuration templates is relative to the <...>_metadata.xml file. For example, <TemplateFile path=" ./Templates/<...>.xml" /></p> <p>Complete path: This must include the product identity of add-in (an environment variable) as part of the path and that can be obtained from the add-in tool. For example, <TemplateFile path="\$OPEN.ABB.CRB1100SAFETYTEMPLATES/Safety/Templates/<...>.xml" />. OPEN.ABB.CRB1100SAFETYTEMPLATES is the product identity.</p>

All tags and arguments should always look like the following example:

```
<?xml version="1.0" encoding="utf-8" ?>
<Templates version="1.0" xmlns
  ="urn:abb-robotics-registry-safety-templates">
  <Template name="Template_1">
    <Title text="User-friendly text"/>
    <Description text="Description for the template"/>
    <Date>When it is created (Date and Time)</Date>
    <TemplateFile path=" ./Templates/<...>.xml" />
  </Template>
  <Template name="Template_2">
    <Title text="User-friendly text"/>
    <Description text="Description for the template"/>
    <Date>When it is created, (Date and Time)</Date>
    <TemplateFile path=" ./Templates/<...>.xml" />
  </Template>
</Templates>
```

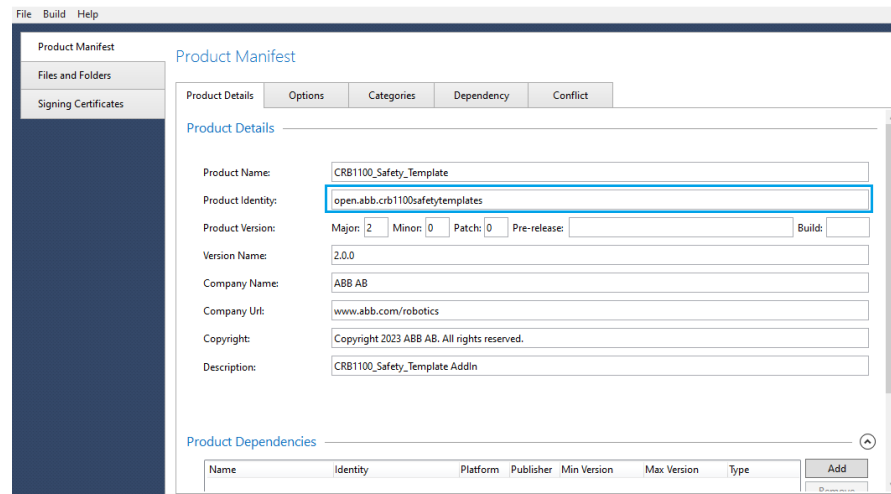
Continues on next page

2 Reference material

2.2 Register safety template

Continued

For the complete path (`<TemplateFile path=" " />`) for the safety configuration template, product identity environment variable shall be obtained from the add-in tool as shown in the following figure.



The screenshot shows the 'Product Manifest' tool with the 'Product Details' tab selected. The 'Product Identity' field is highlighted with a blue border and contains the text 'open.abb.crb1100safetytemplates'. Other fields include Product Name (CRB1100_Safety_Template), Product Version (Major: 2, Minor: 0, Patch: 0, Pre-release: , Build:), Version Name (2.0.0), Company Name (ABB AB), Company Url (www.abb.com/robotics), Copyright (Copyright 2023 ABB AB. All rights reserved.), and Description (CRB1100_Safety_Template Addin). The Product Dependencies section is also visible at the bottom.

xx2300001521

Templates

`Templates` is not a mandatory directory, but it is good to create this directory to keep all the safety templates in one location. This directory shall contain all the safety configuration templates, those can be distributed and installed into the RobotWare system to configure the Safety Controller.

You can add any number of safety configuration templates in this directory. These templates can later be browsed and loaded from the Flexpendant SafeMove app on the Template Selector page onto the Safety Controller.

Display of directory and file structure in the Add-in packaging tool

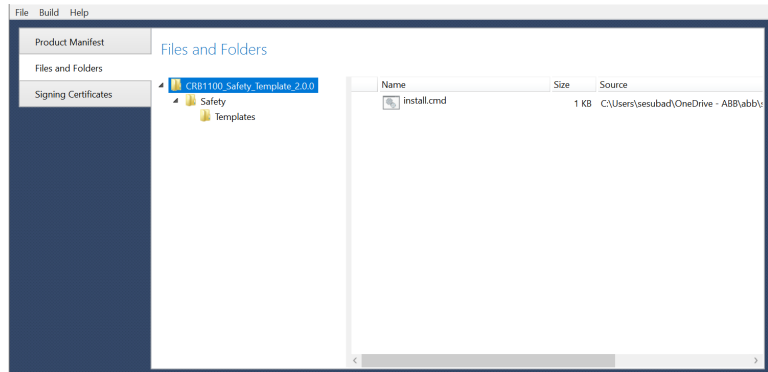
Once the directory structure for the Add-in is ready, the next step is to package everything together and create an Add-in package that can be distributed and installed into the RobotWare system.

The following images display the sample directory and file structure in Add-in Packaging Tool.

Continues on next page

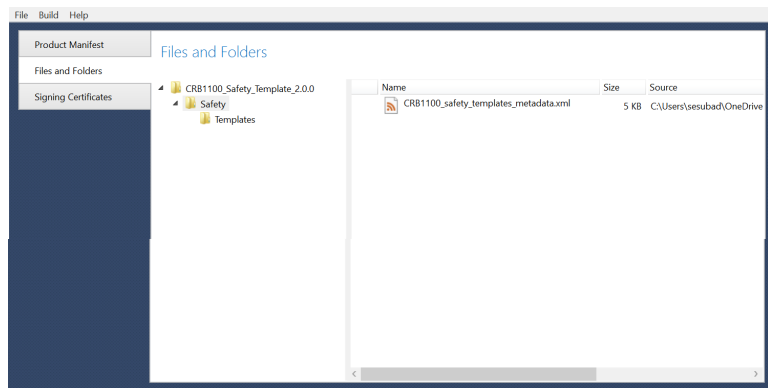
The content of an Add-in is developed in the **Files and Folders** section. Prepare the directory and files for the Add-in to be added so that it can be distributed and installed into a RobotWare system.

- Root Folder



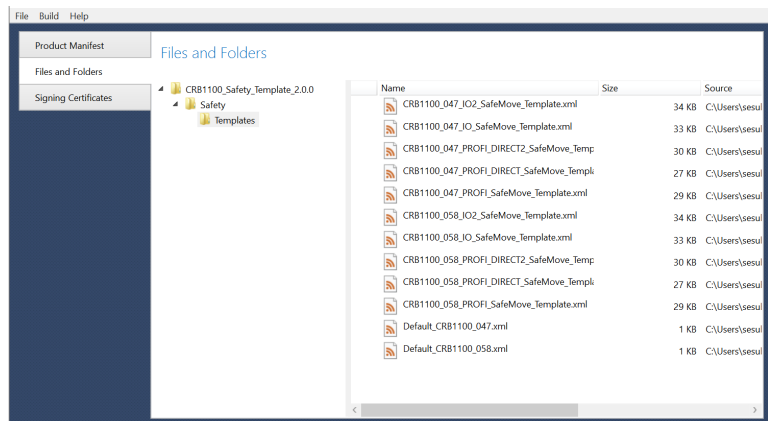
xx2300001527

- Safety



xx2300001528

- Templates



xx2300001529

2 Reference material

2.3.1 About cfg files

2.3 System parameters related to add-in development

2.3.1 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:

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

Example

EIO:CFG_1.0::	Domain EIO without version number
EIO:CFG_1.0:5:0::	Domain EIO with version number 5.0
EIO:CFG_1.0:6:0::	Domain EIO with version number 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

2 Reference material

2.3.1 About cfg files

Continued

```
# -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 "TCPIP1" -Type "TCP/IP" -PhyChannel "LAN1"
```

2.3.2 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*.


Parameter	Description
File	The name of the file including the path on the controller. An environment variable can preferably be used. That is, <code><environment variable>:/<file name></code> . See setenv on page 64 .
Task	Name of a task, if it should only be loaded to one specific task.  Note The parameters <i>Task</i> , <i>Shared</i> , <i>AllTask</i> and <i>AllMotionTask</i> are mutually exclusive.
Shared	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.  Note The parameters <i>Task</i> , <i>Shared</i> , <i>AllTask</i> and <i>AllMotionTask</i> are mutually exclusive.  Note The parameter <i>Shared</i> cannot be combined with <i>Installed</i> .
AllTask	Defines if the module should be loaded into all tasks.  Note The parameters <i>Task</i> , <i>Shared</i> , <i>AllTask</i> and <i>AllMotionTask</i> are mutually exclusive.
AllMotionTask	Defines if the module should be loaded into all motion tasks.  Note The parameters <i>Task</i> , <i>Shared</i> , <i>AllTask</i> and <i>AllMotionTask</i> are mutually exclusive.

Continues on next page

2 Reference material

2.3.2 Topic Controller

Continued

Parameter	Description
Installed	<p>A module can be loaded or installed.</p> <p>A loaded module will behave like a module manually loaded from the teach pendant.</p> <p>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.</p> <p>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.</p> <div> Note</div> <p>The parameter <i>Installed</i> cannot be combined with <i>Shared</i>.</p>
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*.

Parameter	Description
ExcludeFileFromHomeAtBackup	This file in the <i>HOME</i> directory shall not be included in the backup.

Continues on next page

Parameter	Description
ExcludeDirFromHomeAtBackup	This directory in the <i>HOME</i> directory shall not be included in the backup.
IncludeFileAtBackup	This file is not located in the <i>HOME</i> directory, but shall be included in the <i>BACKINFO</i> directory in the backup.
IncludeDirAtBackup	This directory is not located in the <i>HOME</i> directory, but shall be included in the <i>BACKINFO</i> directory in the backup.

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 49](#).

2 Reference material

2.3.3 Topic I/O System

2.3.3 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

```
EIO:CFG_1.0::  
#  
EIO_SIGNAL:  
-Name "DOAccessInternal" -SignalType "DO" -Access "internal"  
-Name "DOAccessInternalRAPID" -SignalType "DO" -Access "internal"  
-Category "rapid"
```

2.3.4 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.

Parameter	Description
name	The title of the custom pick list.
type	The type that contains the instruction names of the pick list

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.

Parameter	Description
name	The name of the instruction.



Note

- The pick list types contains 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

2 Reference material

2.3.4 Topic Man-machine Communication

Continued

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.

Parameter	Description
name	The instruction name.
default_struct	Defines which arguments should have proposed values. <ul style="list-style-type: none">• 0: No proposed value• 1: A proposed value. If alternative arguments, 1 indicates that the first alternative argument should be used with a proposed value.• 2: Only for alternative arguments. The second alternative argument should be used with a proposed value.• 3: Only for alternative arguments. The third alternative argument should be used with a proposed value.• 4: Only for alternative arguments. The fourth alternative argument should be used with a proposed value.
hidden	Defines if the instruction should be hidden when showing RAPID routines. If <i>hidden</i> is set, the instruction will not be shown when choosing an instance for ProcCall or Move PP to Routine. For changes of the <i>hidden</i> parameter to take effect, the controller must be restarted by using the restart mode Reset RAPID or Reset system . A restart is not enough.



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
```

Argument	Argument number	Argument alternative
TriggData	1	0
Distance	2	0
Start	3	1
Time	3	2
Interrupt	4	0

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
```

Continues on next page

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*.

Parameter	Description
param_nr	Specifies the argument numbers that should not reuse values from previous instruction calls.

Example

The instruction *MoveL* is defined with the following arguments:

```
MoveL [\Conc] ToPoint [\ID] Speed [\V] | [\T] Zone [\Z] [\Inpos]
Tool [\WObj] [\Corr] [\TLoad]
```

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:

```
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*.


Parameter	Description
name	The instruction argument, defined as <i><instruction name>_<argument name></i> (for example <i>MoveL_Tool</i>). It is also possible to define a common argument name (<i>common_<argument name></i>) to be used in the type <i>MMC_COMMON_PARAM</i> .

Continues on next page

2 Reference material

2.3.4 Topic Man-machine Communication

Continued

Parameter	Description
name_rule	<p>Specifies how the argument proposal should be generated. The following rules can be used:</p> <ul style="list-style-type: none">• NONE - Unexpanded placeholder. No proposal is generated.• CUR - The parameter <i>method</i> 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 <i>def_name</i>.• 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 <i>def_name</i> is used.• VAL - No argument identifier is used. A literal value is used instead.
method	<p>Method to be called if <i>name_rule</i> is CUR or SEQ. Supported methods are:</p> <ul style="list-style-type: none">• 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	<p>Default name needed if <i>name_rule</i> is LAST or DEF.</p> <div> Note</div> <p>A string must have 3 quotation marks:</p> <pre>-name Direction -name_rule LAST -def_name "" "Z" ""</pre>

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`.

Argument	Argument proposal
V	<p>If V is actively selected it should:</p> <ol style="list-style-type: none">1 use the value from the last instruction using V2 use the default value 1000
ID	<p>No identifier should be proposed for ID. A numeric value is proposed instead. The proposed numeric value is defined in <code>MMC_REAL_DATATYPE</code>.</p>
T	<p>If T is actively selected it should use the default value 5.</p>
Z	<p>If Z is actively selected it should:</p> <ol style="list-style-type: none">1 use the value from the last instruction using Z2 use the default value 50
Tool	<p>The proposal for Tool should be defined by the method <code>hirule_tooldata</code>.</p>
WObj	<p>The proposal for WObj should be defined by the method <code>hirule_wobjdata</code>.</p>

Continues on next page

Argument	Argument proposal
TLoad	The proposal for TLoad should be defined by the method <code>hirule_tloaddata</code> .
common_point	The proposal for common_point should: 1 be a sequential increase from the last robtarget 2 be defined by the method <code>hirule_robtarg</code>
common_speed	The proposal for Tool should: 1 use the value from the last instruction using <code>speeddata</code> 2 use the default value 1000
common_zone	The proposal for common_zone should: 1 use the value from the last instruction using <code>zonedata</code> 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_robtarg
-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.

Parameter	Description
name	The instruction argument, defined as <i><instruction name>_<argument name></i> (for example <i>MoveL_Tool</i>).
common_space_name	Name of the common argument defined in <i>MMC_REAL_PARAM</i> .

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

2 Reference material

2.3.4 Topic Man-machine Communication

Continued

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*.

Parameter	Description
name	Name of the data type.
def_name	Default base identifier for the data (for example tool). The identifier for the data is created from the <i>def_name</i> 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).
value_rule	Specifies how the value of the new data should be generated: <ul style="list-style-type: none">• NONE - No initialize value for non-value data type.• CUR - The parameter <i>method</i> is used to define the data value. For example used when a robtarget is given the value of the current robot TCP.• DEF - The data value should be a default value defined by the parameter <i>use_value</i>.• 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 <i>use_value</i>. If no data is found, when looking up to 100 statements back, a zero value is used.
method	Method to be called if <i>value_rule</i> is CUR. Supported methods are: <ul style="list-style-type: none">• hirule_robtarget - current robot TCP robtarget value• hirule_jointtarget - current robot TCP jointtarget value• hirule_tooldata - current tooldata value• hirule_wobjdata - current wobjdata value• hirule_tloaddata - current tload value
use_value	Default value if <i>value_rule</i> is DEF or SEQ. Also used as increment value if <i>value_rule</i> is SEQ.
object_type	Data object type (i.e. CONST, VAR, PERS or TASK PERS).
validate_hook	Method to be called when validating data. Supported methods are: <ul style="list-style-type: none">• hirule_validate_tooldata• hirule_validate_wobjdata• hirule_validate_robtarget• hirule_validate_orient• hirule_validate_pose• hirule_validate_progdisp• hirule_validate_loaddata

Example

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

Data type	Proposed data value
identno	If no <i>identno</i> exists, the value is 10. Otherwise the value from the last <i>identno</i> is increased with 10.

Continues on next page

Data type	Proposed data value
robtarget	The new <code>robtarget</code> gets the value of the current robot TCP. A validation is used so that the value of a <code>robtarget</code> cannot be changed to an incorrect format.

```
MMC_REAL_DATATYPE:
-name identno -def_name id -value_rule SEQ -use_value 10 \
-object_type CONST
-name robtarget -def_name p -value_rule CUR \
-method hirule_robtarget -object_type CONST\
-validate_hook hirule_validate_robtarget
```

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.

Parameter	Description
param_nr	Parameter number for the argument to be selected.

Example

The instruction `MoveC` is defined with the following arguments:

```
MoveC [\Conc] CirPoint ToPoint [\ID] Speed [\V] | [\T] Zone [\Z]
[\Inpos] Tool [\WObj] [\Corr] [\TLoad]
```

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.

Parameter	Description
name	Name of the instruction.
param_nr	Argument number for the <code>\WObj</code> optional argument.

Example

```
MMC_INSTR_WITH_WOBJ:
-name MoveL -param_nr 10
```

Continues on next page

2 Reference material

2.3.4 Topic Man-machine Communication

Continued

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.

Parameter	Description
name	Name of the instruction.
param_nr	Argument number for the \TLoad optional argument.

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.

Parameter	Description
name	Name of the instruction.
param_nr	Argument number for the circular point, CirPoint.

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.

Parameter	Description
name	The instruction argument, defined as <instruction name>_<argument name> (for example MoveL_Tool).

Example

The instruction MToolTCPCalib is defined with the following arguments:

```
MToolTCPCalib 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 MToolTCPCalib_Pos1
-name MToolTCPCalib_Pos2
-name MToolTCPCalib_Pos3
-name MToolTCPCalib_Pos4
```

Continues on next page

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.

Parameter	Description
name	The name of the data type.

Example

```
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.

Parameter	Description
name	The name of the instruction.
param_nr	Identifies the optional argument.
use_if_present	Defines if the offset shall be considered if the argument is present (1) or when it is not present (0).

Example

```
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.

Parameter	Description
name	The instruction argument, defined as <i><instruction name>_<argument name></i> (for example <i>MoveC_CirPoint</i>).

Example

```
MMC_NO_PC_MOVEMENT:
-name movec_cirpoint
```

Continues on next page

2 Reference material

2.3.4 Topic Man-machine Communication

Continued

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.

Parameter	Description
name	The instruction argument, defined as <code><instruction name>_<argument name></code> .

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.

Parameter	Description
name	Instruction name.

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`.

Parameter	Description
name	Name of the existing instruction.
shift_name	Name of the instruction it should be changed to.
shift_mode	Motion mode of instruction after changing motion mode.
param_restr	Defines an argument number. If this argument is set, change of motion is not allowed.

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
```

2.3.5 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 67](#).

```
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_robtarg
-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_wobjdata

#
```

Continues on next page

2 Reference material

2.3.5 Example cfg files

Continued

```
MMC_INSTR_WITH_WOBJ:
```

```
-name MoveCircle -param_nr 6
```

2.4 The install.cmd file

2.4.1 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 the **Modify Installation** function) 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

Continues on next page

2 Reference material

2.4.1 Introduction

Continued

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 65](#) and [setstr on page 65](#).

Predefined script variables are the following:

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

- **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:

Environment variable name	ReadOnly/Read-Write	Intended usage
HOME	RW	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.
DATA	RW	Directory intended for logs or similar files that should not be included in backups, and which are created from RAPID programs.

Continues on next page

Environment variable name	ReadOnly/Read-Write	Intended usage
BACKUP	RW	Directory intended for storing backups.
TEMP	RW	Directory for storing temporary files. This directory is cleaned on system Reset (I-Start) and during system updates.
RAMDISK	RW	Directory located in RAM disk (non-persistent) for high-performance logging. Content is lost on each restart of the controller.
XXX.YYY.ZZZ (e.g. OPEN.ABB.ROBOT-ICS.CIRCLEMOVE)	RO	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.
XXX.YYY.ZZZ_HOME (e.g. OPEN.ABB.ROBOT-ICS.CIRCLEMOVE_HOME)	RW	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.
XXX.YYY.ZZZ_DATA (e.g. OPEN.ABB.ROBOT-ICS.CIRCLEMOVE_DATA)	RW	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.

In addition to the system pre-defined environment variables, add-in developers can define custom variables from their installation scripts (see [setenv on page 64](#)). 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.

2.4.2 Commands

addintvar

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

Parameter	Description	Default
name	Name of the script variable.	
value	Value to add.	

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

or

- Generate one line - "include" command into a script to make it include another script.

Parameter	Description	Default
from	Name of an existing file or script that shall be appended or included.	
to	Path of the file in which the content shall be appended to.	
paste	If FALSE, then "include" statement shall be generated in the script specified by "to" argument (append by reference). If TRUE, then contents of file "from" shall be appended to the file "to" (append by value).	FALSE

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.

Parameter	Description	Default
path	Name of an existing file or directory that shall be modified.	
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"
```

Continues on next page

2 Reference material

2.4.2 Commands

Continued

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.

Parameter	Description	Default
path	Name of XML file containing type definition.	

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.

Parameter	Description	Default
name	Path of XML file containing type definition. Types can currently only be created in the PROC domain of the configuration database.	

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 non-internal instances
- Write-protect instances or protect instances from deletion

Parameter	Type	Description	Default
filename	string	The full <i>cfg</i> file name, including the path of the file to load.	

Continues on next page

Parameter	Type	Description	Default
internal	boolean	Write-protect the <i>cfg</i> instances, defined in the <i>cfg</i> file. The write-protected instances will not be included when <i>cfg</i> 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, see Exceptions on page 55 .	FALSE
replace	boolean	All instances defined in the <i>cfg</i> file will replace existing non-internal instances with the same name in the <i>cfg</i> domain. To create a new instance and keep all modified values, the user has to define all attributes and their values in the <i>cfg</i> file. If any attribute is not defined in the file, the default value will be used. See Exceptions on page 55 .	FALSE
modify	boolean	Modify the attribute values, defined in the <i>cfg</i> file, in existing named non-internal instances. Values of other attributes which are not included in the <i>cfg</i> file will remain the same.	FALSE
nondeletable	boolean	Protect the <i>cfg</i> instances, defined in the <i>cfg</i> file, from deletion.	FALSE
force	boolean	Affects only if used together with the "-replace and -internal" arguments. Same behavior as for "-replace and -internal" except that existing internal/write-protected instances also will be replaced. Replaced instances will be write-protected (internal). See Exceptions on page 55 .	FALSE

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 && -force) > (-replace && -internal) > -internal > -replace

Exceptions

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

The first combination results in a replace of existing non-internal instances and the replaced instances will also get write-protected (internal). Already existing internal/write-protected instances will not be replaced, they will be ignored.

The second combination results in a replace of existing non-internal/internal instances and the replaced instances will also get write-protected (internal).

Examples



Note

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

Continues on next page

2 Reference material

2.4.2 Commands

Continued

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
```

Replace existing instances (also if internal/write-protected) and add new instances, all replaced/new instances will be write-protected (internal):

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

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"
```

Continues on next page

copy

Copy a file.

Parameter	Description	Default
from	The file to be copied, including the file path.	
to	The new file name, including the file path.	

Example:

```
copy -from $BOOTPATH/instop.cmd -to $RWTEMP/instop.cmd
```

delay

Delay the running of the command script.

Parameter	Description	Default
time	Number of milliseconds to delay.	100

Example:

```
delay -time 1000
```

delete

Delete a file.

Parameter	Description	Default
name	Name of file to delete, including file path.	

Example:

```
delete -path $RWTEMP/opt_10.cmd
```

direxist

If a directory exists, go to a label.

Parameter	Description	Default
path	The complete path to the folder.	
label	The label to go to if the folder exists.	

Example:

```
direxist -path $TEMP/MyFolder -label CLEANUP_0
```

echo

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

Parameter	Description	Default
text	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.	
elog	Adds an internal event log with the text as message.	FALSE

Examples:

```
echo -text "Installing configuration files"
```

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

Continues on next page

2 Reference material

2.4.2 Commands

Continued

fileexist

If a file exists, go to a label.

Parameter	Description	Default
path	File name, including the file path.	
label	The label to go to if the file exists.	

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.

Parameter	Description	Default
path	File to search, including the file path.	
find	String to find.	
replace	String to replace with.	

Example:

```
find_replace -path $HOME/myfile.txt -find "ABC" -replace "CBA"
```

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.

Parameter	Description	Default
id	Name of the key whose value is to be retrieved.	
strvar	Name of the variable where the result (the key value) is stored.	
errlabel	Label to go to if an error occurs.	

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`.

Parameter	Description	Default
strvar	A string containing the label name to go to.	
label	Label to go to	

Examples:

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

Continues on next page

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.

Parameter	Description	Default
id	The ID of the feature that should be checked.	
robot	The robot number to check if the feature is present. Useful in multimove systems.	1
label	Label to go to if the feature is present.	

Example:

```
if_feature_present -id
abb.robotics.robotcontrol.options.multitasking -robot 1 -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.

Parameter	Description	Default
name	Name of the script variable.	
value	Integer value to compare to.	
label	Label to go to if values are equal.	

Example:

```
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.

Parameter	Description	Default
strvar	String variable to be compared with a string value.	
value	String value to compare the string variable with.	
label	Label to go to if the comparison is true.	

Example:

```
ifstr -strvar $ANSWER -value "IRT5454_2B" -label APP2
```

Continues on next page

2 Reference material

2.4.2 Commands

Continued

ifvc

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

Parameter	Description	Default
label	Label to go to if the script is run on a virtual controller.	

Example:

```
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.


Parameter	Description	Default
path	The file name of the included script, including the file path.	

Example:

```
include -path $BOOTPATH/instdrv.cmd
```

install_io_project

The `install_io_project` command can be used to install I/O project files from an add-in so the controller will start up with all the needed I/O configurations.

Parameter	Type	Description	Default
internal	boolean	Write-protect the <i>CFG</i> instances, defined in the <i>ioeprj</i> file. The write-protected instances will not be included when <i>CFG</i> 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.  Note The argument <i>-internal</i> does not apply to the safety related parts of an I/O project.	FALSE



Note

For configurations with **Safety Config Status** set to **Locked**, the existing configuration cannot be overridden using the `install_io_project` command.

Examples

Add I/O projects without write-protection:

```
install_io_project -file $BOOTPATH/config/myproject.ioeprj
```

Add I/O projects with write-protection:

```
install_io_project -file $BOOTPATH/config/myinternalproject.ioeprj  
-internal
```

Continues on next page

loop_break

Used to break execution of loop_include.

No parameters.

Example:

See [loop_include on page 61](#).

loop_include

Used to execute a script in a loop.

Parameter	Description	Default
path	Path of the script to execute in loop.	
cycles	Maximal number of times to execute the specified script.	

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.

Parameter	Description	Default
size	The size in bytes.	20000 bytes

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.

Parameter	Description	Default
path	Directory name, including the path.	

Example:

```
mkdir -path $RWTEMP/newdir
```

Continues on next page

2 Reference material

2.4.2 Commands

Continued

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.

Parameter	Description	Default
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: <ul style="list-style-type: none">• <i>goto</i> - Go to a label• <i>continue</i> - Ignore errors and continue execution• <i>stop</i> - Stop execution of startup task using <i>assert()</i>• <i>sysfail</i> - Call <i>SYS_FAIL()</i>• <i>return</i> - 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 <i>goto</i> .	

Examples:

```
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.

Parameter	Description	Default
text	The text to show on the console.	

Example:

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

rapid_delete_palette

Deletes a picklist in the FlexPendant programming window.

Parameter	Description	Default
palette	The name of the picklist to be deleted.	

Example:

```
rapid_delete_palette -palette "M.C 3"
rapid_delete_palette -palette "Settings"
```

Continues on next page

rapid_delete_palette_instruction

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

Parameter	Description	Default
palette	The name of the picklist.	
instruction	The name of the RAPID instruction to be deleted.	

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*.

Parameter	Description	Applies to type
type	Defines which type (for example elogmes, option, rapid_metadata, or rapid_text) that is being registered.	
domain_no	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.	elogmes
min	The first message number in the file being registered.	elogmes
max	The last message number in the file being registered.	elogmes
prepath	The path to the language directory.	elogmes, elogrules, rapid_metadata
postpath	The rest of the path, after the language directory, including the character \ (backslash) and the file name.	elogmes, elogrules, rapid_metadata
extopt	A flag indicating that the add-in is an external add-in.	option

Continues on next page

2 Reference material

2.4.2 Commands

Continued

Parameter	Description	Applies to type
description	The name of the add-in.	option
path	The path to the add-in.	option

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.

Parameter	Description	Default
from	Path of the existing original file or directory.	
to	New name.	

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 49](#).

Parameter	Description	Default
name	The environment variable to be assigned a new value.	
value	The string to assign to the environment variable.	

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.

Parameter	Description	Default
name	Name of a new or an existing script variable.	
value	Value to set.	

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.

Parameter	Description	Default
strvar	The string variable to be assigned a new string.	
value	The string to assign to the string variable.	
envvalue	The name of the environment variable. Its string value will be assigned to the string variable.	

Examples:

```
setstr -strvar $LANG -value "en"  
setstr -strvar $CFGPATH -value $SYSPAR  
setstr -strvar $MY_SCRIPT_VAR -envvalue "MY_ENV_VAR"
```

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 70](#).

Parameter	Description	Default
filename	Name of the description file, including the file path.	
package	Package for building the text resource.	"en"

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.

Continues on next page

2 Reference material

2.4.2 Commands

Continued

xattrib

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

Parameter	Description	Default
path	Name of a file directory to modify.	
attrs	Attributes to modify: -R – remove write protection from a file or directory +R – apply write protection on a file or directory	

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.

Parameter	Description	Default
from	Name of a directory to copy.	
to	Location to copy to.	
create_dest_dir	If specified, the destination directory shall be created if it does not exist.	FALSE

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.

Parameter	Description	Default
path	Name of a directory to delete.	
unprotect	Remove write protection if necessary.	FALSE

Examples:

```
xdelete -path $TEMP/MyDir
xdelete -path $TEMP/MyDir2/*
```

2.4.3 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

2.5.1 RAPID modules

2.5 RAPID

2.5.1 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

2.5.2 Using text resources from files

2.5.2 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  
goto -label END  
#INSTALL_FILE  
text -filename $BOOTPATH/language/$LANG/CircleMove_text.xml -package  
$LANG  
#END
```

2.5.3 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 33](#).

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)

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

ENDPROC
ENDMODULE
```

Continues on next page

2 Reference material

2.5.3 Hiding RAPID content

Continued

CircleMoveImpl.sys - Implementation

```
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 37](#).

Example

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"
```

2.5.4 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 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 63](#)) or should be named *rapid_edit_rules.xml* and installed in the \$(HOME) directory of the controller.

Continues on next page

2 Reference material

2.5.4.1 Hiding arguments in programs

2.5.4.1 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 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 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>
```

Continues on next page

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 43](#), or additional optional argument in pick lists, see [Pick list titles \(MMC_PALETTE_HEAD\) on page 37](#). For the latter case the argument will be added, but it will not be shown.

2 Reference material

2.5.4.2 Hiding optional argument when changing selected instruction

2.5.4.2 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 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 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

show	showeditor	Comment
<not defined>	<not defined>	Default, same as True, True
True	True	Shown everywhere in FP
True	False	Hidden in Program Editor and Production Window

Continues on next page

2.5.4.2 Hiding optional argument when changing selected instruction

Continued

show	showeditor	Comment
False	True	Hidden in Argument Window , but shown in Program Editor and Production Window . 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.
False	False	Totally hidden, cannot be edited by Program Editor

2 Reference material

2.5.4.3 Argument filter

2.5.4.3 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 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:

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

Continues on next page

Examples

Some examples:

- The expression "MoveL" (or "^MoveL\$") 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 Reference material

2.5.4.4 Argument value range

2.5.4.4 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 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.

2.6 RobotWare Add-In Packaging tool

2.6.1 Introduction

2.6.1.1 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 using the **Modify Installation** function. 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 **Modify Installation** function.
- Define one or more optional features and rules for how options can be selected in the **Modify Installation** function.
- 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 **Modify RobotWare** function in RobotStudio.

For more information about the **Modify Installation** function, 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 using the **Modify Installation** function to be able to install the add-in.

Continues on next page

2 Reference material

2.6.1.1 About the RobotWare Add-In Packaging tool

Continued

Installation procedure

Before installing the software make sure that the certificates are available, for more information see [Digital signing on page 86](#).

- 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.

2.6.1.2 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

2 Reference material

2.6.1.2 Optional features

Continued

For example, PROFinergy requires that either PROFINET Controller/Device or PROFINET Device is selected for installation:

```
963-1 PROFinergy
<OR dependent on>
888-2 PROFINET Controller/Device
888-3 PROFINET Device
```

2.6.1.3 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 **Modify Installation** function.
- 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 using the **Modify Installation** function. 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.

For more information about product manifest files, see [Appendix: Product manifest files guidelines \(RobotWare 7\) on page 119](#).

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 Reference material

2.6.1.4 Signing with digital certificates

2.6.1.4 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.)

Continues on next page

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**.

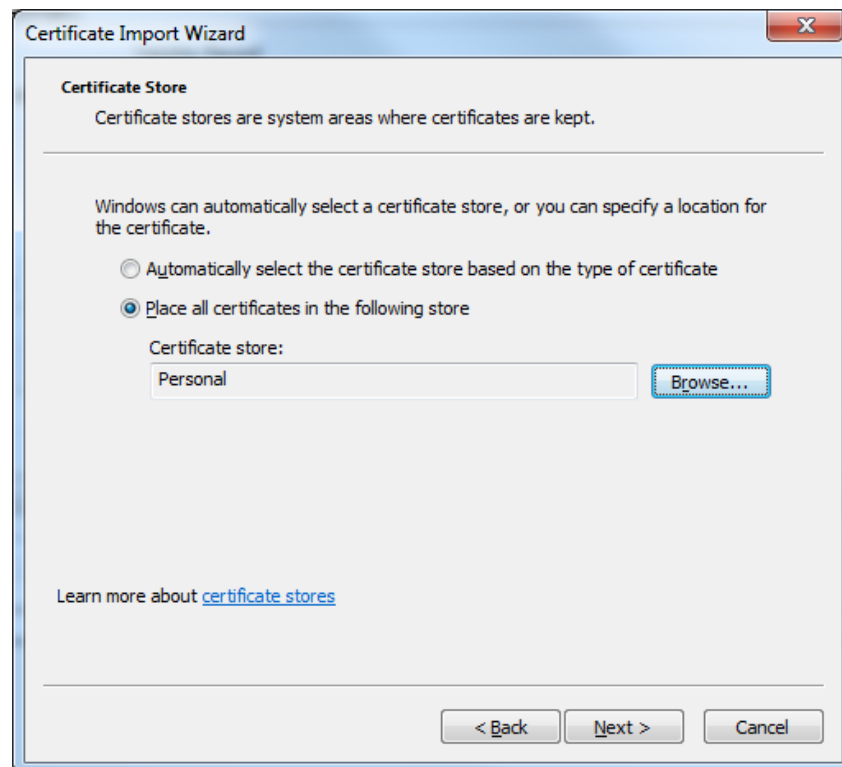
Continues on next page

2 Reference material

2.6.1.4 Signing with digital certificates

Continued

Proceed through the wizard (you will need the pfx password provided by ABB) until prompted to select a certificate store:



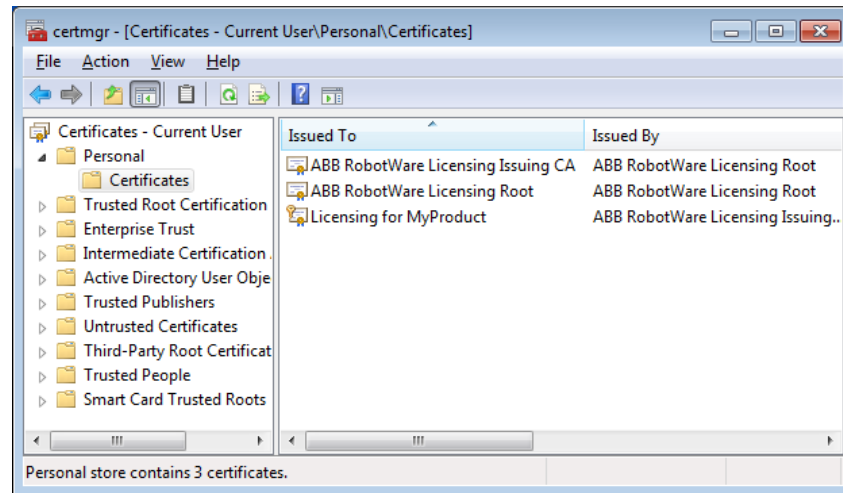
xx1500000935

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.

Continues on next page

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*.



xx1500000936

2 Reference material

2.6.1.5 Types of add-in packaging tools

2.6.1.5 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 91](#) and [Building an add-in from the console on page 107](#).

2.6.2 User interface

2.6.2.1 The home page

The home page

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

Project1 - RobotWare Add-In Packaging tool (C:\My RobotWare Addins\Project1)

File Build Help

Product Manifest

Files and Folders

Signing Certificates

Product Manifest

Product Details Options Categories Dependency Conflict

Product Details

Product Name: YourProductName

Product Identity: open.yourcompany.yourproduct

Product Version: Major: 1 Minor: 0 Patch: 0 Pre-release: Build:

Version Name: 1.0.0

Company Name: company name

Company Url: company url

Copyright: Copyright 2020 company name. All rights reserved.

Description: YourProductName Addin

Product Dependencies

xx2000001995

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.

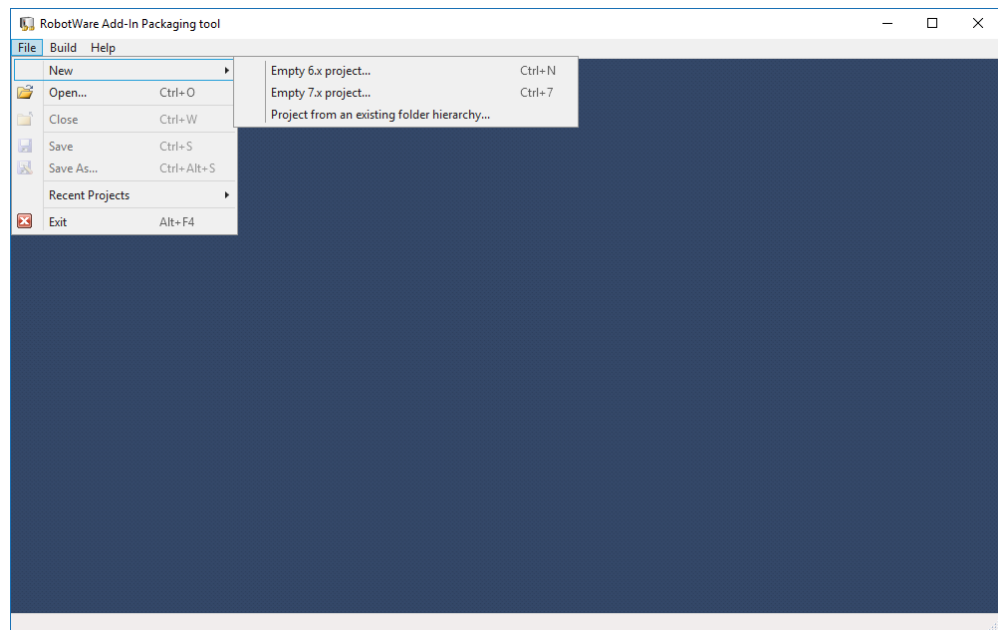
2 Reference material

2.6.2.2 The File menu

2.6.2.2 The File menu


The File menu

The **File** menu is used to manage the projects:




xx2000001996

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

Name	Description
New	<p>Creates a new add-in project. The following options are available:</p> <ul style="list-style-type: none">• 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. <p>For details about creating a project, see Creating and building an add-in project on page 106.</p>
Open	<p>Opens an existing add-in project.</p> <div> Note</div> <p>The add-in project file extension is .rpkproj</p>
Close	<p>Closes the current active project.</p>
Save	<p>Saves the current active project.</p>

Continues on next page

Name	Description
Save As	<p>Save the current active project to a different location on the file system/network.</p> <div> Note</div> <p>The Save As operation saves all the project details in project files (.rp-kproj, .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.</p>
Recent Projects	Displays a list of 10 recently closed projects. You can choose to open a recent project directly, instead of using the Open menu item.
Exit	Exits the tool.

2 Reference material

2.6.2.3 The Product Manifest view

2.6.2.3 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

Product Details

Product Name: YourProductName

Product Identity: open.yourcompany.yourproduct

Product Version: Major: 1 Minor: 0 Patch: 0 Pre-release: Build:

Version Name: 1.0.0

Company Name: company name

Company Url: company url

Copyright: Copyright 2020 company name. All rights reserved.


Description: YourProductName AddIn

Product Dependencies



xx2000001995

Product Details tab

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

Field name	Description
Product Name	The name of the product.
Product Identity	<p>The internal identifier that uniquely identifies the product.</p> <ul style="list-style-type: none">For licensed products the Product Identity must start with one of the namespace strings defined by the licensing certificate. For more information, see Digital signing on page 86.For unlicensed products the Product Identity must start with the string <i>open</i>, for example: <i>open.yourcompany.yourproduct</i>. <p> Note</p> <p>The namespace must be unique and may not contain the id of another product. If so, it will not be possible to select both products when creating an RS system.</p> <p>For example, if product A has id <i>open.mycompany.A</i>, then product B <i>cannot</i> have id <i>open.mycompany.A.B</i>.</p> <p>The id must be changed to <i>open.mycompany.A_B</i> or some other unique name.</p>

Continues on next page

Field name	Description
Product Version	<p>The product version field is used to uniquely identify a specific build of the product. This information is used by the Modify Installation function and other tools to determine the relation between different releases of a product, that is, older, equal, or newer.</p> <p>The format of the product version follows the standards for Semantic Versioning 2.0.0: <code><Major>.<Minor>.<Patch>.<Pre-release>.<Build></code></p> <p> Note</p> <p>The format differs between RobotWare 6 and 7.</p>
Version Name	<p>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 "<i>Beta</i>" or "<i>Release Candidate</i>" in addition to the version.</p> <p>For example, if the Product Version is "<i>2.1.0-32.Internal.Beta1+32</i>" the Version Name can be "<i>2.1.0-32.Internal.Beta1</i>".</p> <p> Note</p> <p>Add-Ins built with version 1.3 or older of the RobotWare Add-In Packaging tool are displayed in the Modify Installation function as a combination of the Product Name and Product Version fields.</p> <p>From version 1.4 the Version Name is used instead of Product Version and it is therefore important that this field contains relevant information.</p>
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 <i>Add-In</i> .

Continues on next page

2 Reference material

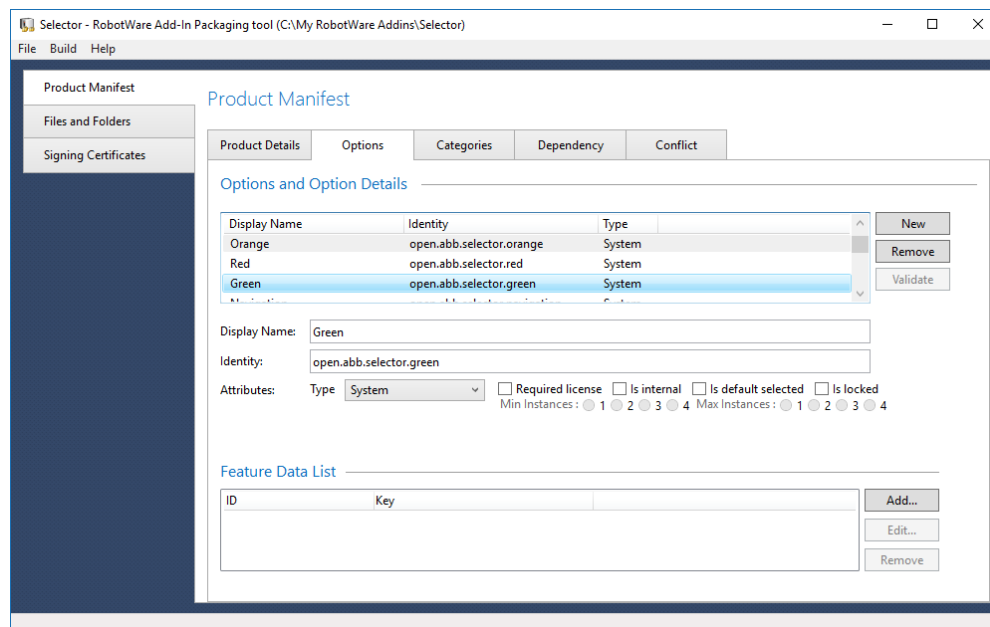
2.6.2.3 The Product Manifest view

Continued

Options tab

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.






xx2000001998

The information is to be defined in the following fields:

Field name	Description
New	Displays all fields that must be completed for the creation of a new option.
Remove	Removes the selected option.
Validate	Validates the newly created option.
Display Name	Type the name of the option. This name is displayed in the Modify Installation function in RobotStudio.
Identity	Type the internal identifier of the option. This id is what uniquely identifies an option in a product. The identifier must begin with the the internal identifier of the product. For example: open.yourcompany.yourproduct.youroption For more information, see Optional features on page 83 .
Type	Select the type of the option: <ul style="list-style-type: none">• 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.

Continues on next page

Field name	Description
Attributes	<p>Select the option attributes:</p> <ul style="list-style-type: none"> • Required license - The option requires a license. • Is internal - The option is not shown in the Modify Installation function user interface. • Is default selected - The option is selected by default in the Modify Installation function in RobotStudio. • Is locked - The option cannot be deselected in Modify Installation function in RobotStudio. <p> Note</p> <p>For licensed products, at least one option should have the Required license check box selected.</p>
Min Instances	<p>The minimum number of robots in the system that can have the option.</p> <p> Note</p> <p>This field is only valid for the option type <i>Robot</i>.</p>
Max Instances	<p>The maximum number of robots in the system that can have the option. For example, when using several robots in a MultiMove system.</p> <p> Note</p> <p>This field is only valid for the option type <i>Robot</i>.</p>

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 58](#).

Field name	Description
Id	Id of the key value.
Key	Key value.

Continues on next page

2 Reference material

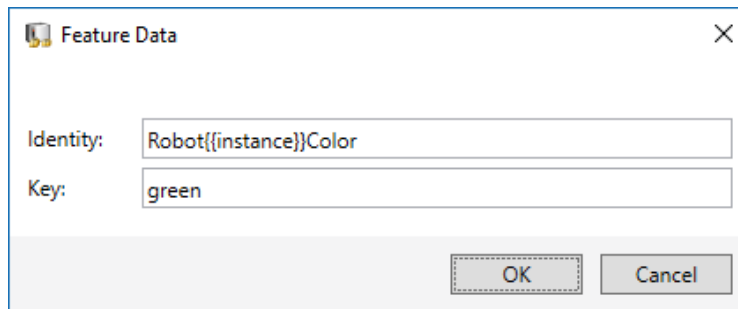
2.6.2.3 The Product Manifest view

Continued

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`.



The screenshot shows a dialog box titled "Feature Data" with a close button (X) in the top right corner. Inside the dialog, there are two text input fields. The first field is labeled "Identity:" and contains the text "Robot{{instance}}Color". The second field is labeled "Key:" and contains the text "green". At the bottom right of the dialog, there are two buttons: "OK" and "Cancel".

xx2000001999

During installation, the **Modify Installation** function 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:

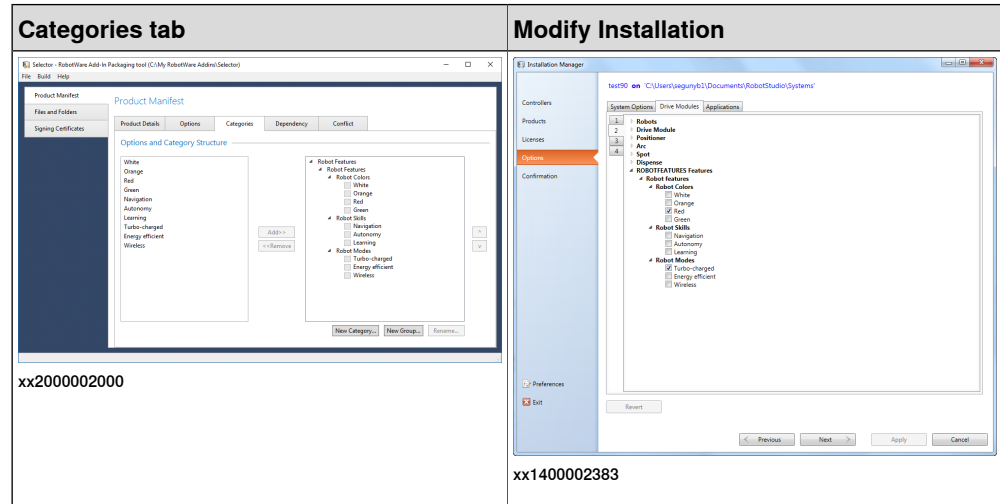
```
getkey -id "ROBOT1COLOR" -var 10 -strvar $ANSWER -errlabel ERROR
goto -strvar $ANSWER
#ORANGE
config ...
#NEXT
#ERROR
```

Continues on next page

Categories tab

The **Categories** tab is used to group and structure the options according to how the add-in should be displayed in the **Modify Installation** user interface.

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.

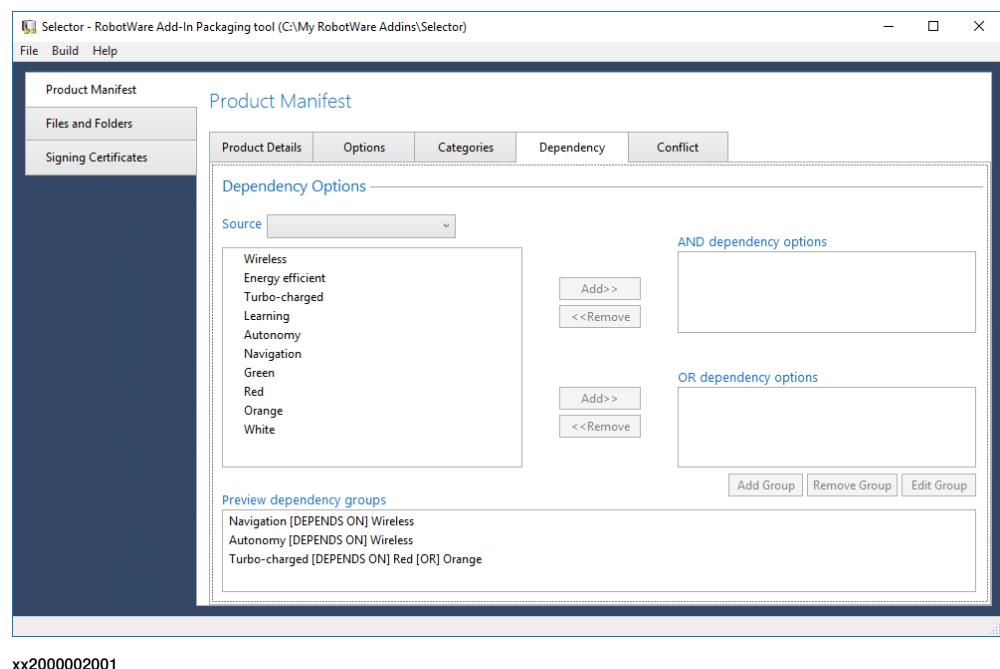


The following validation is performed:

- 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.



Continues on next page

2 Reference material

2.6.2.3 The Product Manifest view

Continued

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 83](#).



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.



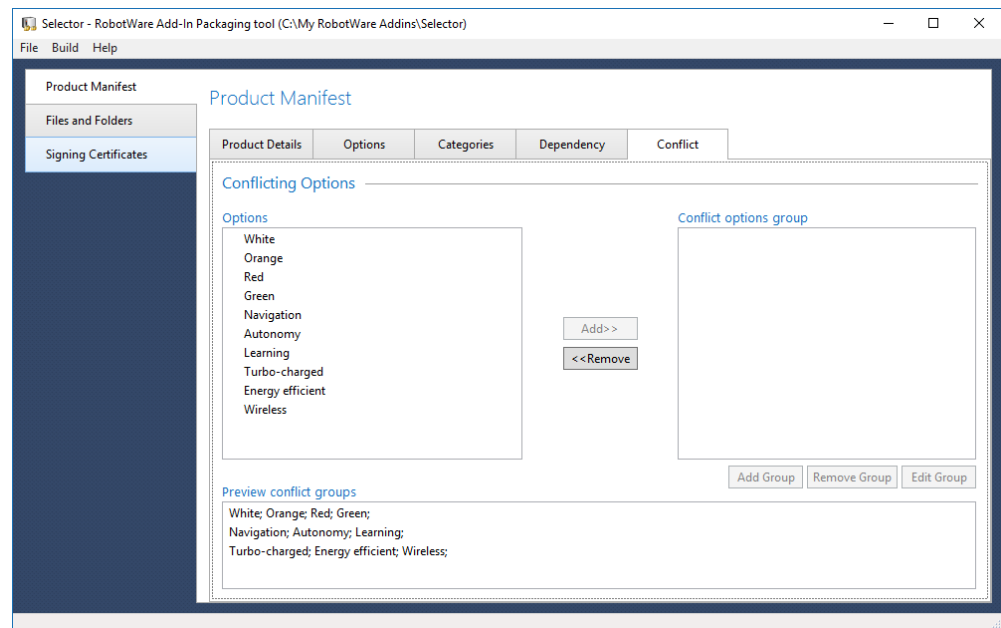
Note

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.

Continues on next page

Conflict tab

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



xx2000002002

By configuring the conflicts, the conflicting options cannot be selected at the same time in the **Modify Installation** user interface.

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.

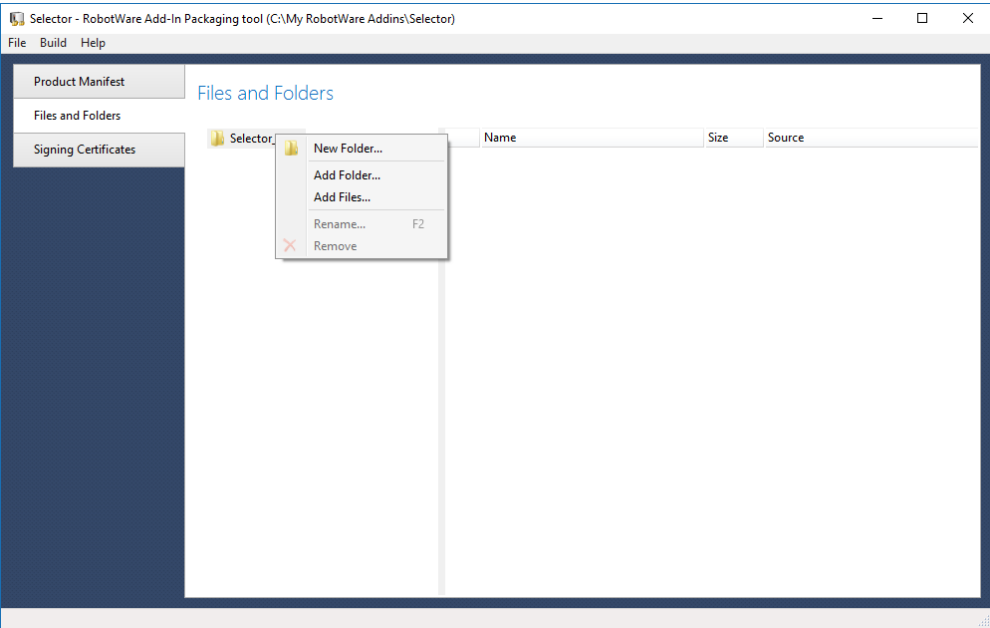
Files and folders can be added to the project using the **Files and Folders** view.

2 Reference material



2.6.2.3 The Product Manifest view

Continued

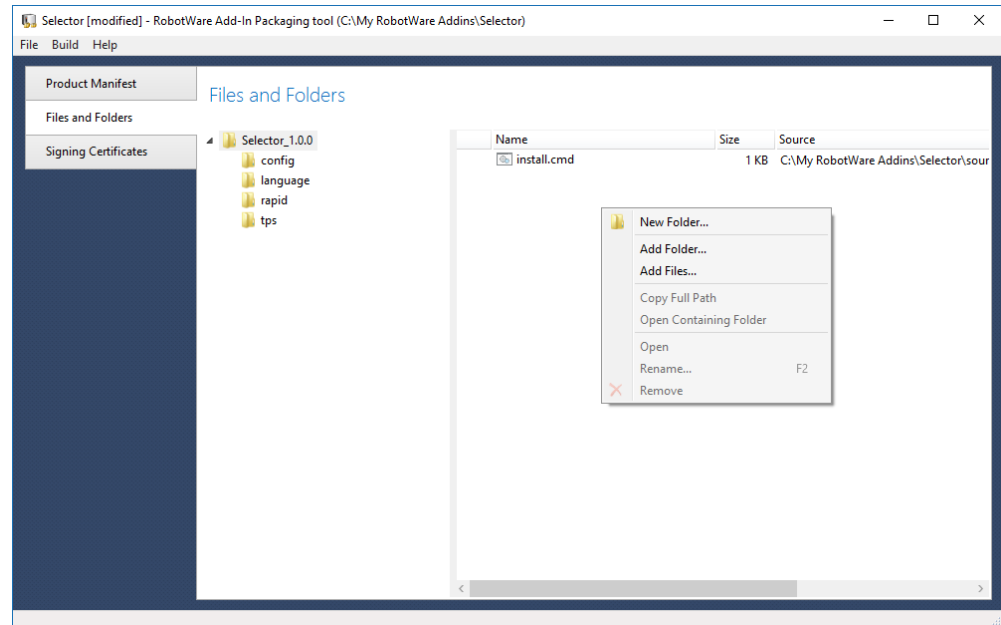
Right-click at the folder level for the following options:



xx2000002003

Field name	Description
New Folder	Creates a new folder. The folder is added to the respective level on the tree.
Add Folder	Adds an existing folder on the file system to the project.
Add Files	Adds the individual files to the project.
Rename	<div>Renames the selected folder.</div> <div> Note The root folder cannot be renamed.</div>
Remove	<div>Removes the selected folder.</div> <div> Note The root folder cannot be removed.</div>

Right-click inside a folder for the following options:



xx2000002004

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

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.

2 Reference material

2.6.2.3 The Product Manifest view

Continued

- 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.

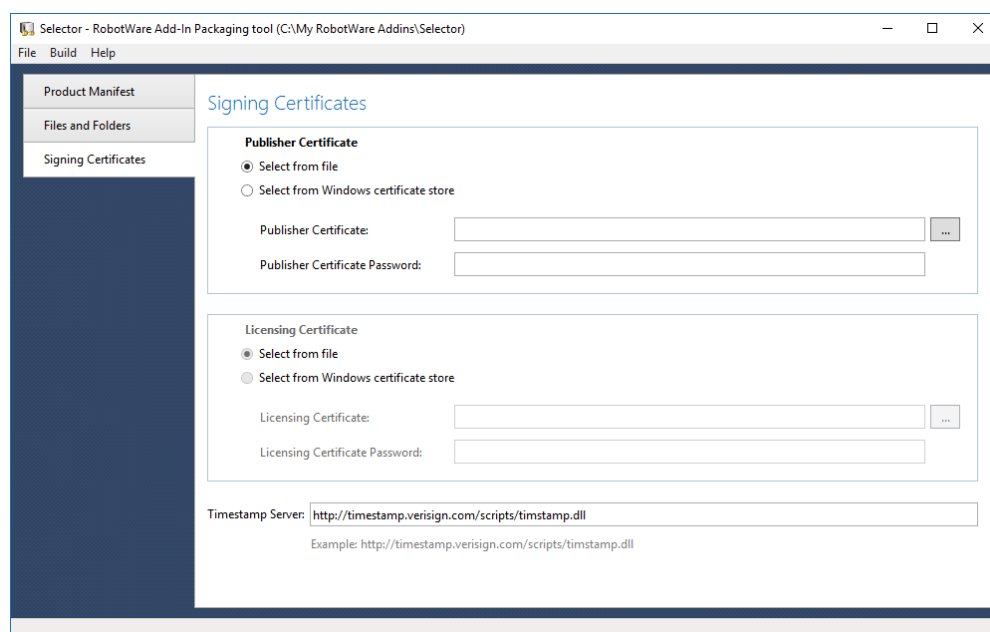
Files and folders for converted add-ins

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 86](#).



The screenshot shows the 'Selector - RobotWare Add-In Packaging tool' window. The left sidebar has three tabs: 'Product Manifest', 'Files and Folders', and 'Signing Certificates'. The 'Signing Certificates' tab is selected. The main area is titled 'Signing Certificates' and contains two sections: 'Publisher Certificate' and 'Licensing Certificate'. Both sections have radio buttons for 'Select from file' (selected) and 'Select from Windows certificate store'. Below each section are input fields for the certificate name and password, with a browse button (three dots) next to the certificate name field. At the bottom, there is a 'Timestamp Server' input field with the example URL 'http://timestamp.verisign.com/scripts/timestamp.dll'.

xx2000002005

Section	Field name	Description
Publisher Certificate	Select from file	<p>Select this option if the publisher certificate for digital signing should be provided as <i>.pfx</i> files.</p> <p> Note</p> <p>Browse Publisher Certificate to select the certificate from its stored location.</p>
	Select from Windows certificate store	Select this option if the publisher certificate for digital signing should be installed on your PC from the Windows certificate store.
	Publisher Certificate	<p>Browse to select a certificate (<i>.pfx</i>file) from its stored location. The selected path is displayed.</p> <p> Note</p> <p>This field is used in combination with option Select from file.</p>
	Publisher Certificate Password	The password for the publisher certificate when specified as a <i>.pfx</i> file.
Licensing Certificate	Select from file	<p>Select this option if the licensing certificate for digital signing should be provided as <i>.pfx</i> files.</p> <p> Note</p> <p>Browse Licensing Certificate to select the certificate from its stored location.</p>
	Select from Windows certificate store	Select this option if the licensing certificate for digital signing should be installed on your PC from the Windows certificate store.
	Licensing Certificate	<p>Browse to select a certificate (<i>.pfx</i>file) from its stored location. The selected path is displayed.</p> <p> Note</p> <p>This field is used in combination with option Select from file.</p>
	Licensing Certificate Password	The password for the licensing certificate when specified as a <i>.pfx</i> file.
	Timestamp Server	<p>Displays the URL to a timestamp server.</p> <p>For more information, see Timestamping on page 86.</p>

2.6.3 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 94](#) 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 101](#) 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 104](#) 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 109](#) for more details.
- 7 Verify the add-in by building a system using the **Modify Installation** function in RobotStudio. See *Operating manual - RobotStudio* for more information.

2.6.4 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:

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

Continues on next page

2 Reference material

2.6.4 Building an add-in from the console

Continued

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.

2.7 License Generator

2.7.1 Introduction

General

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

Installation procedure



Note

The License Generator, including the license certificate, must be ordered from ABB.

- 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

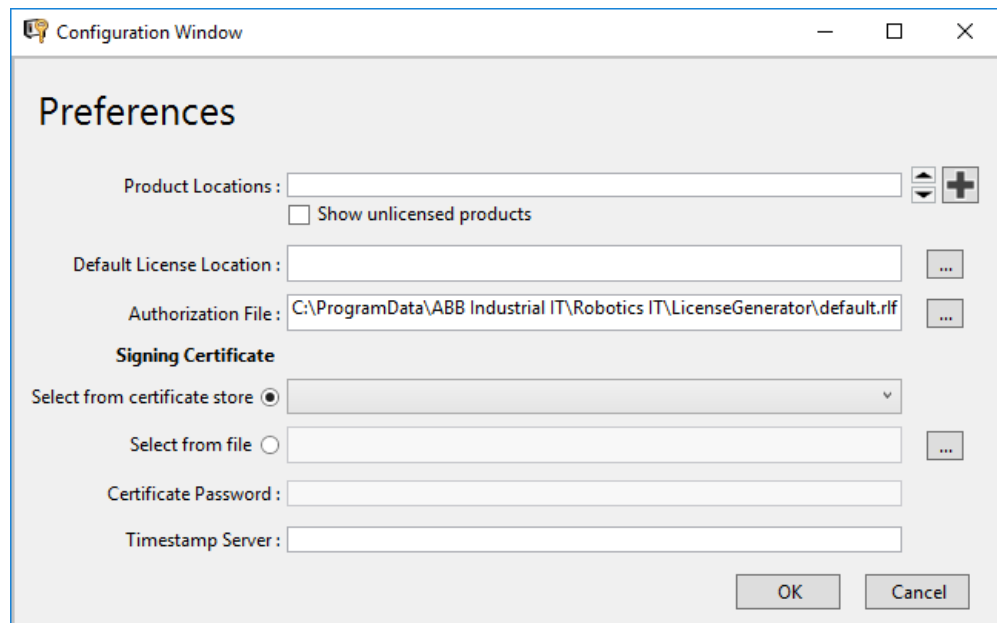
2.7.2.1 The Preferences window

2.7.2 The user interface

2.7.2.1 The Preferences window

Preferences

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



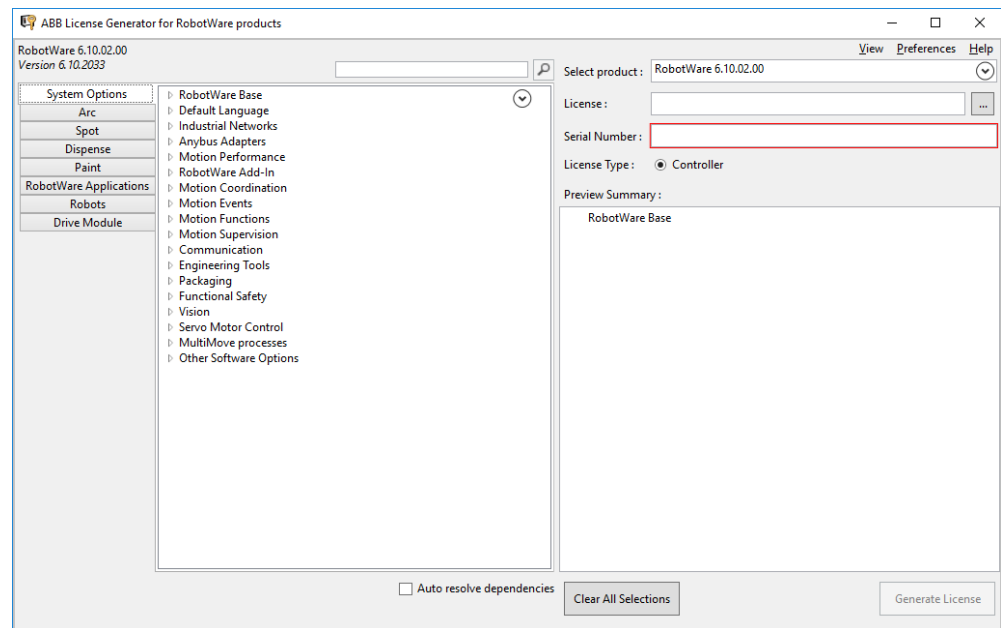
xx2000002041

Field name	Description
Product Locations	The location of the product manifest files (*.rmf).
Default License Location	The default location where the created licence files (*.rlf) should be saved.
Authorization File	The authorization file, license file, (*.rlf) for the License Generator provided by ABB.
Signing Certificate (radio button)	Install/use the certificate provided by ABB, same certificate as for the RobotWare Add-In Packaging tool. <ul style="list-style-type: none">• Select from certificate store - Select certificates from store if the certificates are already installed.• Select from file - Select certificates from file (*.pfx) to install the certificates.
Certificate Password	Use the certificate password provided by ABB.
Timestamp Server	URL to a timestamp server. For more information, see Timestamping on page 86 .

2.7.2.2 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**.



xx2000002040

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



Tip

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

Continues on next page

2 Reference material

2.7.2.2 The main window

Continued



Tip

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

2.7.3 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 110](#).

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 110](#).
- 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 licence.
- 6 Click **Generate License** to generate the license file.
- 7 Verify the license by building a system using the **Modify Installation** function 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 110](#).

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 110](#).
- 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 **Modify Installation** function 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

2 Reference material

2.7.3 Creating the license

Continued

- 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 49](#). The product installation directory is read-only and add-ins cannot remove the write protection. See environment variables under [Introduction on page 49](#) 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.

Name	Migration status	Description
addvar	Replaced	Replaced by addintvar on page 53 .
append	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
attrib	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
config	Not changed	
copy	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
delay	Not changed	
delete	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
delete_cfg_instance	Not available	Currently not available in RW 7.1. Can be considered for RW 7.2. Use cases are requested for this command.
direxist	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
echo	Not changed	
fileexist	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
find_replace	Available with restrictions	Path restrictions apply. See Introduction on page 49 .

Continues on next page

Name	Migration status	Description
getkey	Not changed	Do not use argument <code>-varno</code> . Consider using command if_feature_present on page 59 .
goto	Not changed	
if	Replaced	Replaced by ifintvar on page 59 .
ifstr	Not changed	
ifvc	Not changed	
include	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
invoke	Removed	Available in RW 6, but not documented and supported.
mkdir	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
loop	Replaced	Replaced by loop_include on page 61 and loop_break on page 61 .
onerror	Not changed	
print	Not changed	
rapid_delete_palette	Not changed	
register	Not changed	
rename	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
setenv	Not changed	
setstr	Not changed	
setvar	Replaced	Replaced by setintvar on page 65 .
text	Not changed	
timestamp	Not changed	
uas_install_application_grants	Not available	Currently not available in RW 7.1. Can be considered for RW 7.2. Uses cases are requested for this command.
uas_install_groups	Not available	Currently not available in RW 7.1. Can be considered for RW 7.2. Uses cases are requested for this command.
xattrib	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
xcopy	Available with restrictions	Path restrictions apply. See Introduction on page 49 .
xdelete	Available with restrictions	Path restrictions apply. See Introduction on page 49 .

Continues on next page



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.

This page is intentionally left blank

B Appendix: Product manifest files guidelines (RobotWare 7)

Overview

This appendix contains a list of additional guidelines that should be followed as best practices when structuring product manifest files in add-in packaging tool. Following these guidelines will ensure that the add-in is properly displayed in RobotStudio when modifying RobotWare system configuration of a real and virtual controller that utilizes the add-in.

General product level guidelines

Unique product ID

Each product needs to have a unique product ID that is not a subset of another product ID.

Example:

The following two product IDs overlap and will not be handled correctly when configuring a system that includes both products:

```
open.mycompany.productA
open.mycompany.productA.extension
```

Views

Multiple appearances of a feature

For those parts of the view tree structure that can be visible at the same time, it is possible but not recommended to expose the same feature in multiple places in the view.



Note

A feature has always the same value regardless of where it appears in the view.



Note

Option/feature editor hides child nodes of unselected nodes.

Build hierarchies

Use `<FeatureRef>` element in the view:

- For those branches in the view that represent exclusive/conflicting choices , or
- When a view branch and all its child features are optional

Continues on next page

In other cases, it is most likely simpler to just use a `<Group>` element for representing the branch in the view.



Note

`<FeatureRef>` cannot directly contain `<FeatureRef>`s. To add sub-features under a `<FeatureRef>`, it is always necessary to create a new pair of elements:

```
<Groups><Group>...</Group></Groups>
```

Example:

```
<Group displayName="Collaborative Robots">
  <Groups>
    <Group displayName="CRB 1100">
      <FeatureRef id="abb.robotics.robots.robots.crb1100.4-0_47" />
      <FeatureRef id="abb.robotics.robots.robots.crb1100.4-0_58" />
    </Group>
    <Group displayName="CRB 1300">
      <FeatureRef id="abb.robotics.robots.robots.crb1300.11-0_9" />
      <FeatureRef id="abb.robotics.robots.robots.crb1300.10-1_15" />
      <FeatureRef id="abb.robotics.robots.robots.crb1300.7-1_4" />
    </Group>
    <Group displayName="IRB 14050 (Single arm YuMi)">
      <FeatureRef id="abb.robotics.robots.robots.irb14050.0_5-0_5">
        <Groups>
          <Group displayName="Arm Configuration">
            <FeatureRef id="abb.robotics.robots.robots.arm.irb14050.notype" />
            <FeatureRef id="abb.robotics.robots.robots.arm.irb14050.typea" />
          </Group>
        </Groups>
      </FeatureRef>
    </Group>
    <Group displayName="CRB 15000">
      <FeatureRef id="abb.robotics.robots.robots.crb15000.5-0_95" />
      <FeatureRef id="abb.robotics.robots.robots.crb15000.12-1_27" />
      <FeatureRef id="abb.robotics.robots.robots.crb15000.10-1_52" />
    </Group>
  </Groups>
</Group>
```

xx2300001368

Deep hierarchies

Whenever possible, avoid deep hierarchies in the view.

Conflicts

Selecting feature group member

When one and only one member of a feature group must always be selected by the user, then

- define the feature group using `<Conflict>` element, and
- use `isMandatory="true"` attribute on the `<Conflict>` element.

Continues on next page

**Note**

Members of the Conflict group that is marked with *isMandatory*="true" attribute are visualized by the viewer as radio-buttons.

Example:

```
<Conflicts>
  <Conflict isMandatory="true" displayName="Robot variant">
    <FeatureRef id="abb.robotics.robots.robots.irb*" />
    <FeatureRef id="abb.robotics.robots.robots.crb*" />
    <FeatureRef id="abb.robotics.robots.robots.arm.*" />
  </Conflict>
</Conflicts>
```

xx2300001388

Group conflict

A conflict group can, but it does not have to, correspond to a single <Group> in the view.

Example:

Robot variants have multiple named sub groups in the view , but there is just one conflict definition for all robot variants.

```
<Conflicts>
  <Conflict isMandatory="true" displayName="Robot variant">
    <FeatureRef id="abb.robotics.robots.robots.irb*" />
    <FeatureRef id="abb.robotics.robots.robots.crb*" />
    <FeatureRef id="abb.robotics.robots.robots.arm.*" />
  </Conflict>
</Conflicts>
```

xx2300001389

```
<Category displayName="Robots" type="robot">
  <Groups>
    <Group displayName="Articulated Robots">
      <Groups>...</Groups>
    </Group>

    <Group displayName="Collaborative Robots">
      <Groups>...</Groups>
    </Group>

    <Group displayName="Parallel Robots">
      <Groups>...</Groups>
    </Group>

    <Group displayName="SCARA Robots">
      <Groups>...</Groups>
    </Group>
  </Groups>
</Category>
```

xx2300001390

Dependencies

- Keep dependencies as simple and minimal as possible!
- Do not use circular dependencies between features. If used, behavior is undefined.

Continues on next page

- Only use dependencies between features that are defined in the same manifest file and to features from other products that are listed in the product dependency list
- Avoid using AND and OR dependencies between features that have parent - child relationships in views (applies to both directions: parent -->child and child -->parent)
- Use AND dependencies to automatically pull in dependent features:
 - that are present in different branches of the view tree and between features in the same group of a view (avoid dependencies between parent --> child nodes as mentioned above), or
 - to pull in hidden (internal) features
- Use OR dependencies to refer to a dependent feature group (or a subset of it) that is defined in another branch of the view tree. (Advanced)

Example:

Robot variant > Controller drive system(s)

Licensing

Licence protected features

Features that are license protected can be declared as such in two different ways:

- Directly - those feature definitions directly marked by using `isLicensed="true"` attribute, or
- Indirectly - by making a feature AND dependent to another feature that has attribute `isLicensed="true"`

Hierarchical licensing models

Hierarchical licensing models can be implemented by:

- first defining a set of directly licensed features which represent licensing levels (e.g., Basic, Premium, Premium Plus)
- define ordering of the licensing levels by making them AND dependent to each other (e.g., Basic < Premium < Premium Plus)
- and then assigning each functional feature to one of those license levels by making it "AND dependent" on the chosen licensing level (e.g., "Feature A" > Basic, "Feature A++" > Premium).

Default selection

When possible, define one of conflicting features choices as default. See information about the *isMandatory* attribute in [Selecting feature group member on page 120](#).

Index

A

addintvar, 53
append, 53
attrib, 53

C

CAB_TASK_MODULES, 33
cfg_create_type_from_rules_def, 54
cfg_create_type_from_xml, 54
config, 54
copy, 57

D

default argument values, 37
delay, 57
delete, 57
direxist, 57

E

echo, 57
eio.cfg, 36
event log messages, 19
event log texts, 20

F

fileexist, 58
find_replace, 58

G

getkey, 58
goto, 58

I

if_feature_present, 59
ifintvar, 59
ifstr, 59
ifvc, 60
include, 60
install_io_project, 60
install.cmd, 53, 67

L

load modules, 33
loop_break, 61
loop_include, 61

M

math_lib_set_mem_size, 61
mkdir, 61
mmc.cfg, 37
module, 68
MoveCircle, 68

N

NOSTEPIN, 68

O

onerror, 62

P

pick list, 37
print, 62

R

rapid_delete_palette, 62
rapid_delete_palette_instruction, 63
RAPID module, 68
RAPID rules, 37
register, 63
rename, 64

S

safety, 9
setenv, 64
setintvar, 65
setstr, 65
sys.cfg, 33
system module, 68

T

template file, 19
text, 65
timestamp, 65

V

validating .xml, 22

X

xattrib, 66
xcopy, 66
xdelete, 66
xml
 validating, 22

**ABB AB****Robotics & Discrete Automation**

S-721 68 VÄSTERÅS, Sweden

Telephone +46 10-732 50 00

ABB AS**Robotics & Discrete Automation**

Nordlysvegen 7, N-4340 BRYNE, Norway

Box 265, N-4349 BRYNE, Norway

Telephone: +47 22 87 2000

ABB Engineering (Shanghai) Ltd.

Robotics & Discrete Automation

No. 4528 Kangxin Highway

PuDong New District

SHANGHAI 201319, China

Telephone: +86 21 6105 6666

ABB Inc.**Robotics & Discrete Automation**

1250 Brown Road

Auburn Hills, MI 48326

USA

Telephone: +1 248 391 9000

abb.com/robotics