ROBOTICS

Application manual

RobotWare Add-Ins

Trace back information:
Workspace 22D version a8
Checked in 2022-12-09
Skribenta version 5.5.019

Application manual

RobotWare Add-Ins

RobotWare 6.14.01

Document ID: 3HAC051193-001

Revision: K

© Copyright 2015-2022 ABB. All rights reserved.
Specifications subject to change without notice.

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 2015-2022 ABB. All rights reserved.
Specifications subject to change without notice.

Table of contents

Overview of this manual 7
Product documentation 9
Safety 11

1 Introduction 13

1.1 About RobotWare Add-Ins 13
1.2 The CIRCLEMOVE example 15

2 RobotWare Add-In functionality 17

2.1 Required files and file structure 17
2.2 version.xml 19
2.3 The install.cmd file 21
2.3.1 Introduction 21
2.3.2 Commands 22
2.3.3 Examples of install.cmd files 32
2.4 RAPID modules 33
2.5 Custom event log messages 35
2.5.1 About event log messages 35
2.5.2 Event log texts 36
2.5.3 Event log titles 38
2.5.4 Validating event log .xml files 39
2.6 System parameters related to add-in development 40
2.6.1 About cfg files 40
2.6.2 Topic Controller 43
2.6.3 Topic I/O System 47
2.6.4 Topic Man-machine Communication 48
2.6.5 Example cfg files 58
2.7 Using text resources from files 60
2.8 Hiding RAPID content 62
2.9 Optional settings for RAPID arguments (RAPID meta data) 64
2.9.1 Hiding arguments in programs 65
2.9.2 Hiding optional argument when changing selected instruction 67
2.9.3 Argument filter 69
2.9.4 Argument value range 71
2.10 FlexPendant applications 72

3 RobotWare Add-In Packaging tool 73

3.1 Introduction 73
3.1.1 About the RobotWare Add-In Packaging tool 73
3.1.2 Optional features 75
3.1.3 Files of a packaged add-in 77
3.1.4 Signing with digital certificates 78
3.1.5 Types of add-in packaging tools 82
3.2 User interface 83
3.2.1 The home page 83
3.2.2 The File menu 84
3.2.3 The Product Manifest view 86
3.3 Creating and building an add-in project 97
3.4 Converting an additional option to an add-in 98
3.5 Building an add-in from the console 99

4 License Generator 101

4.1 Introduction 101
4.2 The user interface 102
4.2.1 The Preferences window 102

© Copyright 2015-2022 ABB. All rights reserved.

4.2.2 The main window 103
4.3 Creating the license 105

Index **107**

© Copyright 2015-2022 ABB. All rights reserved.

Overview of this manual

About this manual

This manual contains instructions for how to create your own Add-In to use with ABB's robot systems.

Usage

With the help of this manual, you can package functionality into an Add-In and create a license that allows access to the Add-In.

Who should read this manual?

This manual is intended for:

- line builders that want to implement the same program solution on many robots
- ABB's partners, selling the robot systems with their own functionality added
- ABB companies selling robot systems

Prerequisites

The reader should...

- be experienced in working with ABB robots
- be experienced RAPID programmer
- be familiar with system parameters

References

Reference	Document ID
Technical reference manual - System parameters	*3HAC050948-001*
Operating manual - RobotStudio	*3HAC032104-001*

Revisions

Revision	Description
-	Released with RobotWare 6.00.01 First release.
A	Released with RobotWare 6.02 • Added section about I/O signals, see *Topic I/O System on page 47*. • Updated the path to the utility folders throughout the manual, for example see *Template files on page 35*. • Added section *Hiding RAPID content on page 62*. • Added section *Optional settings for RAPID arguments (RAPID meta data) on page 64* and updated section *register on page 28*. • Updated the section *RobotWare Add-In Packaging tool on page 73*. • Minor corrections.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Continued

Revision	Description
B	Released with RobotWare 6.03 • Added the tag *VersionName* to the *version.xml* file, see *version.xml on page 19*. • Updated the section *register on page 28*. • Added sections *Argument filter on page 69* and *Argument value range on page 71*. • Updated the section *Product Details tab on page 86*. • Minor corrections.
C	Released with RobotWare 6.04 • Added commands for deleting items in the FlexPendant programming window picklist, see *The install.cmd file on page 21*. • Added command direxist, see *Commands on page 22*. • Minor corrections.
D	Released with RobotWare 6.07 • Added the section *Building an add-in from the console on page 99*.
E	Released with RobotWare 6.08 • Updated the section *User interface on page 83*. • Updated the RAPID code example in the section *RAPID modules on page 33*.
F	Released with RobotWare 6.10. • Updated with information for Unicode support, see *Unicode characters in UI and TP instructions in RAPID on page 60*.
G	Released with RobotWare 6.11. • Added more information about arguments in the Unicode support, see *Unicode characters in UI and TP instructions in RAPID on page 60*.
H	Released with RobotWare 6.12. • Updated the section *RobotWare Add-In Packaging tool on page 73*. • New script math_lib_set_mem_size added in section *Commands on page 22*.
J	Released with RobotWare 6.13. • Section *Argument Name Rules (MMC_REAL_PARAM) on page 50* updated with information about how to add a string in Rapid rules.
K	Released with RobotWare 6.14.01. • Updated the section *Building an add-in from the console on page 99*. • Updated the section *Commands on page 22*.

© Copyright 2015-2022 ABB. All rights reserved.

Product documentation

Categories for user documentation from ABB Robotics

The user documentation from ABB Robotics is divided into a number of categories. This listing is based on the type of information in the documents, regardless of whether the products are standard or optional.



Tip

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

Product manuals

Manipulators, controllers, DressPack/SpotPack, and most other hardware is delivered with a **Product manual** that generally contains:

- Safety information.
- Installation and commissioning (descriptions of mechanical installation or electrical connections).
- Maintenance (descriptions of all required preventive maintenance procedures including intervals and expected life time of parts).
- Repair (descriptions of all recommended repair procedures including spare parts).
- Calibration.
- Troubleshooting.
- Decommissioning.
- Reference information (safety standards, unit conversions, screw joints, lists of tools).
- Spare parts list with corresponding figures (or references to separate spare parts lists).
- References to circuit diagrams.

Technical reference manuals

The technical reference manuals describe reference information for robotics products, for example lubrication, the RAPID language, and system parameters.

Application manuals

Specific applications (for example software or hardware options) are described in **Application manuals**. An application manual can describe one or several applications.

An application manual generally contains information about:

- The purpose of the application (what it does and when it is useful).
- What is included (for example cables, I/O boards, RAPID instructions, system parameters, software).
- How to install included or required hardware.
- How to use the application.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Continued

- Examples of how to use the application.

Operating manuals

The operating manuals describe hands-on handling of the products. The manuals are aimed at those having first-hand operational contact with the product, that is production cell operators, programmers, and troubleshooters.

© Copyright 2015-2022 ABB. All rights reserved.

Safety

Safety of personnel

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

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

Safety regulations

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

© Copyright 2015-2022 ABB. All rights reserved.

This page is intentionally left blank

1 Introduction

1.1 About RobotWare Add-Ins

What is an Add-In

In RobotWare 6 the concept of additional options has been replaced with RobotWare Add-Ins. If you were familiar with the additional option concept, you will see that from a structural point of view, additional options and Add-Ins are handled the same way on the robot controller. What is new is that the packaging has been changed to simplify installation with Installation Manager. For more information about the packaging tool and process, see *RobotWare Add-In Packaging tool on page 73*.

Using Add-Ins

Add-Ins allow to create installable supplemental 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 a number of RAPID modules, system modules, or program modules which hold the basic code for the Add-In. The Add-In also includes some files for loading and configuration at start up. The Add-In may also include *.xml* files with event log messages in different languages.

An Add-In can also consist of more advanced coding, such as *C#* code, for FlexPendant applications. This manual will cover the first case, with coding done in RAPID only. For more advanced coding, use RobotStudio SDK applications.

Unlicensed, open, Add-Ins

What you need from ABB to package your own open Add-In is:

- RobotWare Add-In Packaging tool

Licensed Add-Ins

What you need from ABB to package your own licensed Add-In is:

- RobotWare Add-In Packaging tool
- a licence certificate for the RobotWare Add-In Packaging tool for your Add-In name

To license the Add-In, you will also need:

- License Generator
- a publisher certificate.
- a licensing certificate for the License Generator

For more information, see *Digital signing on page 78*.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Basic approach

These are the major steps for creating an Add-In. More detailed descriptions are given later in this manual.

1. Create the RAPID code for the Add-In, see *RAPID modules on page 33*.
2. Create the event message files, see *Custom event log messages on page 35*.
3. Create configuration files for system parameters, see *System parameters related to add-in development on page 40*.
4. Create the file *version.xml*, see *version.xml on page 19*.
5. Create the file *install.cmd*, see *The install.cmd file on page 21*.
6. Create and package the RobotWare Add-In, see *RobotWare Add-In Packaging tool on page 73*.
7. For licensed Add-Ins:

   Create a licence file, see *License Generator on page 101*.
8. Use the Installation Manager in RobotStudio to create a system that uses the Add-In.

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

Selections within Add-Ins

An Add-In can contain selections (that is, optional functionality selectable at installation). For more information see *getkey on page 25* and the **Feature Data** section in the *Options tab on page 88*.

© Copyright 2015-2022 ABB. All rights reserved.

1.2 The CIRCLEMOVE example

Introduction

Throughout this manual, an example Add-In is used to illustrate how to implement an Add-In. This Add-In is called *CIRCLEMOVE*.

Some parts of this manual have detailed reference information for commands and syntax used in the Add-In files. Looking at the examples can be a way of solving your problem without having to read all the reference information.

Description

The Add-In *CIRCLEMOVE* contains an instruction called `MoveCircle` that will move the robot in a complete circle. This instruction is added to a pick list on the FlexPendant and behaves just like one of the original instructions.

Error messages are stored in *.xml* files (in this example only in English), and are used in the RAPID code.

© Copyright 2015-2022 ABB. All rights reserved.

This page is intentionally left blank

2 RobotWare Add-In functionality

2.1 Required files and file structure

Add-In files

An Add-In consists of a number of files that you need to create in order to make your own Add-In.

File type	Description
version.xml	Name, version number and description of the Add-In, see *version.xml on page 19*.
install.cmd	Installation script. Specifies for example which *.cfg* files to load, see *The install.cmd file on page 21*.
.cfg	One or several *.cfg* files with the configuration of system parameters. If the Add-In includes RAPID, one of the *.cfg* files should specify which RAPID module (*.sys* file) to load, see *System parameters related to add-in development on page 40*.
.sys or *.mod*	The RAPID source code, see *RAPID modules on page 33*.

Event log message files

If the Add-In contains customized event log messages, two XML files for each language are required. These are placed in a specific folder for each language:

- *<Add-In folder>\language\<language code>\<Add-In name>_elogtext.xml*
- *<Add-In folder>\language\<language code>\<Add-In name>_elogtitles.xml*

The language codes consist of two letters, for example *en*, *de*, or *fr*, and are defined by the standard ISO 639, see *Custom event log messages on page 35*.

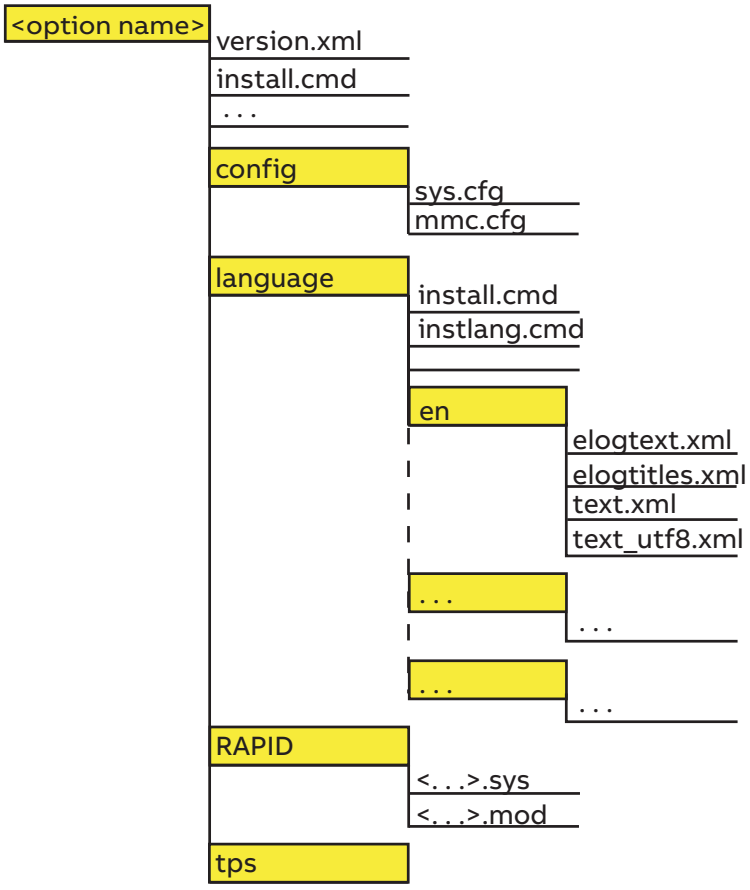
Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

File structure

This picture displays the layout the of an Add-In, required by the robot controller, before it has been packaged with the RobotWare Add-In Packaging tool.

The *tps* folder is used for FlexPendant applications, see *FlexPendant applications on page 72*.



xx1400002781

Note

Note that *mmc.cfg*, *sys.cfg*, and the folder *language* (and everything in that folder) are optional. How to set up the language folders and their content is described in *Including language files from your add-in on page 60*.

© Copyright 2015-2022 ABB. All rights reserved.

2.2 version.xml

Introduction

A RobotWare 6 Add-In contains a *version.xml* file that holds the name, version, and description of the Add-In. This is the same as for a RobotWare 5 additional option.

When converting a RobotWare 5 additional option to a RobotWare 6 Add-In, the information in the *version.xml* file is read by the RobotWare Add-In Packaging tool.

Note

In RobotWare 6, the role of the *version.xml* file has been largely replaced by the product manifest file. However, some client applications may still require the file to be present and the RobotWare Add-In Packaging tool will therefore automatically create a *version.xml* file based on the product manifest information.

XML description

The file *version.xml* is using the following tags:

Tag	Description
Major	Major version number. This number is changed every time a new Add-In version with major changes is released. Integer between 0 and 65535.
Minor	Minor version number. This number is changed every time a new Add-In version with minor changes is released. Integer between 0 and 65535.
Revision	Revision number. This number is changed every time a revision of the Add-In is released. Integer between 0 and 65535.
Build	Build number. To be used internally during developing and testing of the Add-In. Integer between 0 and 65535.
VersionName	The complete product version as displayed to the end user. This may include additional identifiers such as "*Beta*" or "*Release Candidate*".
Title	The name of the Add-In.
Description	A description of what the Add-In is used for. Max 255 characters.
Date	The release date for this version of the Add-In. Format: YYYY-MM-DD
Type	Always set to *AdditionalOption*.

Example file

```
<Version>
  <Major>1</Major>
  <Minor>01</Minor>
  <Revision>01</Revision>
  <Build>001</Build>
```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.2 version.xml
Continued

```
  <VersionName>1.01.01 Beta 2</VersionName>
  <Title>CircleMove</Title>
  <Description>CIRCLEMOVE Add-In (gives access to the instruction
       MoveCircle)</Description>
  <Date>2014-12-31</Date>
  <Type>AdditionalOption</Type>
</Version>
```


© Copyright 2015-2022 ABB. All rights reserved.

2.3 The install.cmd file

2.3.1 Introduction

Description

The install.cmd file is an installation script that for example define which configuration files and event log messages files to load. One of the configuration files (*sys.cfg*) defines which RAPID program files (*.sys*) to load.

© Copyright 2015-2022 ABB. All rights reserved.

2.3.2 Commands

Overview

This section describes the syntax of the commands that can be used in the installation script *install.cmd*.

The script *install.cmd* is executed when using the restart mode **Reset system** to automatically install a number of different files, like configuration files or text files. The *install.cmd* file can contain conditions so that certain actions only are executed if certain conditions are true.

$

$ defines the name of a string variable.

Example:

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

Predefined strings:

String	Predefined value
$BOOTPATH	The folder where *install.cmd* is executed. For example: */hd0a/<system name>/PRODUCTS/<Add-In folder>*
$HOME	*/hd0a/<system name>/HOME*
$SYSPAR	*/hd0a/<system name>/SYSPAR*
$RWTEMP	*/hd0a/temp*

#

Comment, if # followed by a space.

Label, if no space between # and text.

Example:

```
# A comment
#Label
```

config

This command can be used for two different purposes:

- Erase unprotected contents of a configuration domain before loading a new content.

or

- Modify existing contents of a configuration domain.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Erase unprotected contents

The `config` command can be used to erase unprotected contents of a configuration domain before loading a new content.

Parameter	Description	Default
domain	The topic of the *cfg* file. Allowed values are: • *SIO* - Communication • *SYS* - Controller • *EIO* - I/O • *MMC* - Man-machine communication • *MOC* - Motion	
erase	Erase unprotected instances of the specified configuration domain.	FALSE

Example:

```
config -erase -domain MOC
```

Modify existing contents

The `config` command can be used to modify existing contents of a configuration domain.

The following operations are possible:

- Add new instances
- Replace existing non-internal instances
- Modify existing non-internal instances in the specified configuration domain

Parameter	Description	Default
filename	The *cfg* file name, including file path.	
load	The types and instances specified in the *cfg* file will be added to the *cfg* database. If an instance with the same name already exists, an error will be generated and the cfg file will not be loaded. To overwrite the existing instances, use option `-replace`.	TRUE
internal	Load an internal configuration file. All instances loaded with this option will be write-protected. Once they have been loaded, they cannot be overwritten.	FALSE
replace	Replace all existing non-internal instances with the same name as those in the loaded *cfg* file.	FALSE
modify	Modify all existing non-internal instances with the same name as those in the loaded cfg file. With this option it is possible to change one or several parameters of each instance and all other mandatory parameters are not needed.	FALSE

Example:

```
config -filename $BOOTPATH/eio.cfg -load
config -filename $BOOTPATH/sys.cfg -internal
config -filename $BOOTPATH/eiopw.cfg -replace
config -filename $BOOTPATH/awNoSimPrompt.cfg -replace -internal
```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.3.2 Commands
Continued

Note

The cfg files must start with the name of the domain since the config command uses this information to determine the domain. The first row in the cfg file shall 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 and revision
```

Note

When loading a configuration file, only one loading parameter maybe specified at a time (see the examples). The only exception is -replace and -internal that can be used at the same time.

That 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.

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/instopt.cmd -to $RWTEMP/instopt.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_l0.cmd
```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

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 VxWorks console output during the system start.

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

Examples:

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

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.	

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.3.2 Commands
Continued

Parameter	Description	Default
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
```

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
```

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.	

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Example:

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

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
```

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: • *goto* - Go to a label • *continue* - Ignore errors and continue execution • *stop* - Stop execution of startup task using *assert()* • *sysfail* - Call `SYS_FAIL()` • *return* - If used by a script included by another script, execution returns to the calling script. The included script returns an error code that needs to be handled by the including script.	continue
label	The label to go to if action is `goto`.	

Examples:

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

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.3.2 Commands
Continued

print

Prints a text to the VXWorks 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"
```

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 message titles (elogtitle) – register the xml-file to the *elogtext_registry.xml* file.
- Options (option) - Registers the option in the *option_registry.xml* file. This will enable automatic loading of FlexPendant applications from the *tps* folder for the add-in.
- RAPID meta data (rapid_metadata) – Registers additional RAPID argument settings to the *rapid_metadata_registry.xml*.
- RAPID texts (rapid_text) – Registers additional RAPID texts with support for Unicode characters in rapid_text_registry.xml.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

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

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 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 path for RAPID text resource with Unicode support
register -type rapid_text -min 1 -max 123 -resource myAddIn -prepath
     $BOOTPATH/language/ -postpath
myAddInTexts.xml
```

setenv

Define an environment variable and set its value.

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

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

If you define the path to your add-in folder as an environment variable, this variable can be used in your programs instead of hard coding the path.

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
```

System environment variables

The following environment variables are set up by the system and cannot be overwritten.

Environment variable	Value
HOME	*/hd0a/<system name>/HOME*
BACKUP	*/hd0a/BACKUP*
SYSPAR	*/hd0a/<system name>/SYSPAR*
TEMP	*/hd0a/temp*
SYSTEM	*/hd0a/<system name>*
RELEASE	*/hd0a/<system name>/Products/ROBOTWARE_6.XX.XXXX*

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.	

Examples:

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

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 IRC5*, and *Overview on page 60*.

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"
```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

timestamp

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

No parameters.

© Copyright 2015-2022 ABB. All rights reserved.

2.3.3 Examples of install.cmd files

Example for CIRCLEMOVE

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


© Copyright 2015-2022 ABB. All rights reserved.

2.4 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 36*.

```
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;
    p1.trans:=pCenter.trans+[0,Radius,0];
    p2.trans:=pCenter.trans+[Radius,0,0];
```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.4 RAPID modules
Continued

```
    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
```


© Copyright 2015-2022 ABB. All rights reserved.

2.5 Custom event log messages

2.5.1 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.

Two .xml files

Your event log messages are added to the system via two *.xml* files. One *.xml* file contains all the information about the messages. The other one contains their message number and title. Both are required.

These files can be given any name, as long as the installation script *install.cmd* points out the correct file names. It is recommended to use the following names:

- *<Add-In name>_elogtext.xml*
- *<Add-In name>_elogtitles.xml*

Template files

Template files for the two required .xml files are included in the RobotWare installation.

- *template_elogtext.xml*
- *template_elogtitles.xml*

The template files are located in the following directory in the RobotWare installation: *...\RobotPackages\RobotWare_RPK_<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**.

© Copyright 2015-2022 ABB. All rights reserved.

2.5.2 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.

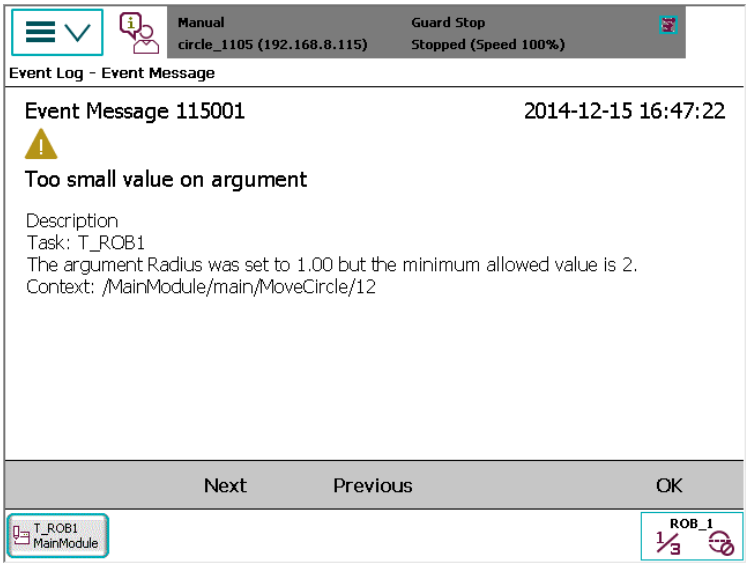
XML tag or argument	Description
domainNo	Event log messages are divided into different domains. Domain number 8 is called *User events* and is reserved for non-ABB messages. For add-ins, always use domain 8 to avoid conflict with messages defined by ABB.
lang	Language code for the text in the messages.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

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

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:



xx1400002871

```
<?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>
```


© Copyright 2015-2022 ABB. All rights reserved.

2.5.3 Event log titles

Overview

For the internal handling of event log messages, the following *.xml* file listing the message numbers and their titles is necessary:

- *<Add-In name>_elogtitles.xml*

Explanation of the .xml file

This is a list of the *.xml* tags and arguments that you need to define. For the complete syntax, see the example below.

XML tag or argument	Description
Title	Create one instance of `Title` for each event log message. The text in the `Title` tag must be identical to the text in the event log text *.xml* file.
number	The same event log message number as in the event log text *.xml* file.

Example of the .xml file

```
<?xml version="1.0" encoding="utf-8"?>
<ExtractTitles>
<Title domain="11" number="5001">Too small value on argument</Title>
</ExtractTitles>
```


© Copyright 2015-2022 ABB. All rights reserved.

2.5.4 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 installation, see *Template files on page 35*.
- The command line tool *XMLFileValidator* can be downloaded from the *RobotStudio Online Community*, where it is included in the *Tools and Utilities* package.

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

```
xmlfilevalidator elogtext.xsd my_elogtext.xml
```

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

Prerequisites

The XMLFileValidator is provided as-is.

Microsoft .NET framework version 2.0 or later is required.

© Copyright 2015-2022 ABB. All rights reserved.

2.6 System parameters related to add-in development

2.6.1 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

© Copyright 2015-2022 ABB. All rights reserved.

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 "COM1" -Connector "COM1"
  -Name "LAN1" -Connector "LAN"
#
```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.6.1 About cfg files

Continued

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


© Copyright 2015-2022 ABB. All rights reserved.

2.6.2 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, *<environment variable>:/<file name>*. See *setenv on page 29*.
Task	Name of a task, if it should only be loaded to one specific task. **Note** The parameters *Task*, *Shared*, *AllTask* and *AllMotionTask* 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 *Task*, *Shared*, *AllTask* and *AllMotionTask* are mutually exclusive. **Note** The parameter *Shared* cannot be combined with *Installed*.
AllTask	Defines if the module should be loaded into all tasks. **Note** The parameters *Task*, *Shared*, *AllTask* and *AllMotionTask* are mutually exclusive.
AllMotionTask	Defines if the module should be loaded into all motion tasks. **Note** The parameters *Task*, *Shared*, *AllTask* and *AllMotionTask* are mutually exclusive.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.6.2 Topic Controller
Continued

Parameter	Description
Installed	A module can be loaded or installed. A loaded module will behave like a module manually loaded from the teach pendant. An installed module will behave like a built in module. By default the attributes NOVIEW and NOSTEPIN are set, even if not stated in the module declaration. Thus it will not be visible from the FlexPendant and can only be removed by using the restart mode **Reset system**. It will not be possible to step into a routine in such a module with FWD. It is recommended that all application modules are installed as built in modules, since then they will be handled as part of the controller and quite separated from the user´s modules. **Note** The parameter *Installed* cannot be combined with *Shared*.
Hidden	RAPID routines and data in this module are hidden from the user.

Example

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

Tip

When loading modules automatically, a correct file path must be used.

Since the name of the directory for the add-in can be changed, the files are often copied to the HOME directory so the file path is unmistakable for "automatic loading of modules".

This can be a problem when doing backup between releases.

Since all files in the HOME directory are saved to the backup, new files copied from the add-in directory will be overwritten by the old files in the backup.

Instead of copying the files to the HOME directory, the files can remain in the add-in directory and therefor avoid copying the files from the HOME directory to the backup.

To access the files in the add-in directory an environment variable must be used, therefore use *setenv on page 29*.

Modules included in a backup

Two things affects what to include when creating a backup:

1. From where the module is loaded.
2. How the configuration file is loaded.

Modules not included in the backup

A module will not be included in the backup:

- if the module is loaded from *$RELEASE*. For example:

  ```
  -File "RELEASE:/options/xxx.modx" -Task "T_ROB1"
  ```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

- if the module is loaded from any user defined environment variable, using *setenv on page 29*.

A module will not be included in the backup and no configuration entries will be listed in the *sys.cfg* (BACKUP/SYSPAR/SYS.CFG):

- if the module is loaded from *$RELEASE*, or any user defined environment variable, and the loaded configuration file is set to *-internal*. For example:

```
config -filename $RELEASE/options/xxx.cfg -domain SYS
-internal
```

Modules included in the backup

A module will be included in the backup:

- if the module is loaded, installed, or shared from elsewhere without *$RELEASE* or any user defined environment variable

A module will be included in the backup but no configuration entries will be listed in the *sys.cfg* (BACKUP/SYSPAR/SYS.CFG):

- if the module is loaded from elsewhere except *$RELEASE* or any user defined environment variable, and the loaded configuration file is set to *-internal*.

  Only loaded modules will be included in the backup, no installed or shared.

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 *HOME* directory shall not be included in the backup.
ExcludeDirFromHomeAtBackup	This directory in the *HOME* directory shall not be included in the backup.
IncludeFileAtBackup	This file is not located in the *HOME* directory, but shall be included in the *BACKINFO* directory in the backup.
IncludeDirAtBackup	This directory is not located in the *HOME* directory, but shall be included in the *BACKINFO* directory in the backup.

Example

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

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.6.2 Topic Controller
Continued

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 21*.

© Copyright 2015-2022 ABB. All rights reserved.

2.6.3 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"
```


© Copyright 2015-2022 ABB. All rights reserved.

2.6.4 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

© Copyright 2015-2022 ABB. All rights reserved.

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. • 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 *hidden* is set, the instruction will not be shown when choosing an instance for ProcCall or Move PP to Routine. For changes of the *hidden* parameter to take effect, the controller must be restarted by using the restart mode **Reset RAPID** or **Reset system**. A restart is not enough.

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

© Copyright 2015-2022 ABB. All rights reserved.

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 *<instruction name>_<argument name>* (for example `MoveL_Tool`). It is also possible to define a common argument name (*common_<argument name>*) to be used in the type `MMC_COMMON_PARAM`.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Parameter	Description
name_rule	Specifies how the argument proposal should be generated. The following rules can be used: • NONE - Unexpanded placeholder. No proposal is generated. • CUR - The parameter *method* is used to define the argument proposal. For example used when the tool argument should use the current tool. • DEF - The argument proposal should be a default value defined by the parameter *def_name*. • SEQ - The argument proposal is based on the previous instruction with a similar argument. Based on the identifier used in the previous instruction, an increment of the index is used to create a new identifier. For example, if the robtarget of the previous move instruction is p10, the next move instruction will propose p20 (unless p20 is already used, then p30, p40, ... will be tried until an identifier is found that is not already used). If no similar argument is found, looking 100 instructions back, a data value is used instead of an identifier. • LAST - The argument proposal gets its value from the previous instruction with a similar argument. If no similar argument is found, looking 100 instructions back, a default value specified by *def_name* is used. • VAL - No argument identifier is used. A literal value is used instead.
method	Method to be called if *name_rule* is CUR or SEQ. Supported methods are: • hirule_robtarget - robtarget symbol name increment value • hirule_jointtarget - jointtarget symbol name increment value • hirule_tooldata - current tooldata • hirule_wobjdata - current wobjdata • hirule_tloaddata - current tload
def_name	Default name needed if *name_rule* is LAST or DEF. **Note** A string must have 3 quotation marks: `-name Direction -name_rule LAST -def_name """Z"""`

Example

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

Argument	Argument proposal
V	If V is actively selected it should: 1 use the value from the last instruction using V 2 use the default value 1000
ID	No identifier should be proposed for ID. A numeric value is proposed instead. The proposed numeric value is defined in *MMC_REAL_DATATYPE*.
T	If T is actively selected it should use the default value 5.
Z	If Z is actively selected it should: 1 use the value from the last instruction using Z 2 use the default value 50
Tool	The proposal for Tool should be defined by the method hirule_tooldata.
WObj	The proposal for WObj should be defined by the method hirule_wobjdata.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.6.4 Topic Man-machine Communication
Continued

Argument	Argument proposal
TLoad	The proposal for TLoad should be defined by the method hirule_tloaddata.
common_point	The proposal for common_point should: 1 be a sequential increase from the last robtarget 2 be defined by the method hirule_robtarget
common_speed	The proposal for Tool should: 1 use the value from the last instruction using speeddata 2 use the default value 1000
common_zone	The proposal for common_zone should: 1 use the value from the last instruction using zonedata 2 use the default value z50

```
MMC_REAL_PARAM:
-name MoveL_V -name_rule LAST -def_name 1000
-name MoveL_ID -name_rule VAL
-name MoveL_T -name_rule DEF -def_name 5
-name MoveL_Z -name_rule LAST -def_name 50
-name MoveL_Tool -name_rule CUR -method hirule_tooldata
-name MoveL_WObj -name_rule CUR -method hirule_wobjdata
-name MoveL_TLoad -name_rule CUR -method hirule_tloaddata

-name common_point -name_rule SEQ -method hirule_robtarget
-name common_speed -name_rule LAST -def_name v1000
-name common_zone -name_rule LAST -def_name z50
```

Argument Identifier Rules (MMC_COMMON_PARAM)

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

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

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

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

© Copyright 2015-2022 ABB. All rights reserved.

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 *def_name* and an index. If nothing else is defined, the index starts at 1 and the increment for each data is 1 (for example the first tooldata is called tool1, the second is called tool2 and so on).
value_rule	Specifies how the value of the new data should be generated: • NONE - No initialize value for non-value data type. • CUR - The parameter *method* is used to define the data value. For example used when a robtarget is given the value of the current robot TCP. • DEF - The data value should be a default value defined by the parameter *use_value*. • SEQ - The data value is based on the previous data of the same data type. The previous value is increased with a value defined by *use_value*. If no data is found, when looking up to 100 statements back, a zero value is used.
method	Method to be called if *value_rule* is CUR. Supported methods are: • 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 *value_rule* is DEF or SEQ. Also used as increment value if *value_rule* 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: • 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 `identno` exists, the value is 10. Otherwise the value from the last identno is increased with 10.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Data type	Proposed data value
robtarget	The new `robtarget` gets the value of the current robot TCP. A validation is used so that the value of a robtarget 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 `\WObj` optional argument.

Example

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

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

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

© Copyright 2015-2022 ABB. All rights reserved.

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 *<instruction name>_<argument name>* (for example MoveC_CirPoint).

Example

```
MMC_NO_PC_MOVEMENT:
-name movec_cirpoint
```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

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 *<instruction name>_<argument name>*.

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
```


© Copyright 2015-2022 ABB. All rights reserved.

2.6.5 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 32*.

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

#
CAB_TASK_MODULES:

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

CircleMove_mmc.cfg

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

```
MoveCircle pCenter Radius Speed Zone Tool [\WObj]
```

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

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

#
MMC_MC1 = MMC_PALETTE:

-name MoveCircle

#
MMC_REAL_ROUTINE:

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

#
MMC_REAL_PARAM:

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

#
```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

```
MMC_INSTR_WITH_WOBJ:

-name MoveCircle -param_nr 6
```


© Copyright 2015-2022 ABB. All rights reserved.

2.7 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 IRC5*.

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 *Required files and file structure on page 17*.

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
```

Unicode characters in UI and TP instructions in RAPID

To support Unicode characters in the text, the registry `rapid_text` is needed. This enables text tables with UTF-8 encoding.

Place text strings in the respective language folder in a file ending with *text_utf8.xml*.

More information about installation of localized language files can be found above in the overview.

The following examples are available in the AddIn Packaging Tool.

Example of how to install/register Unicode files

```
register -type rapid_text -resource circlemove_text -min 200 -max
     206 -prepath $BOOTPATH/language/ -postpath
     /CircleMove_text_utf8.xml
```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Example of how to use/get Unicode in RAPID

To get Unicode text on the FlexPendant, each string resource must be written as:

```
{{<resourceName:textNumber}}
```

Example of resource string to get string 205 in the resource in the AddIn:

```
{{circlemove_text:205}}
```

Use ; (semicolon) as separator to add arguments after the resource string.

```
{{circlemove_text:206;1.00;2}}
```

Strings with arguments in the resource file should be written as below with 1 as start index.

```
The radius {1} is too small, minimum is {2} mm
```

The result on the FlexPendant will be:

```
The radius 1.00 is too small, minimum is 2 mm
```


© Copyright 2015-2022 ABB. All rights reserved.

2.8 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 43*.

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)

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

  ENDPROC
ENDMODULE
```

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

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 48*.

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"
```


© Copyright 2015-2022 ABB. All rights reserved.

2.9 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 28*) or should be named *rapid_edit_rules.xml* and installed in the *$(HOME)* directory of the controller.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.9.1 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

© Copyright 2015-2022 ABB. All rights reserved.

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 54*, or additional optional argument in pick lists, see *Pick list titles (MMC_PALETTE_HEAD) on page 48*. For the latter case the argument will be added, but it will not be shown.

© Copyright 2015-2022 ABB. All rights reserved.

2.9.2 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

© Copyright 2015-2022 ABB. All rights reserved.

2.9.2 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**

© Copyright 2015-2022 ABB. All rights reserved.

2.9.3 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 `s`, where `s` is a sequence of characters. Ranges may be specified as `c-c`.
`[^s]`	Any character not in the set `s`.
`r*`	Zero or more occurrences of the regular expression `r`.
`r+`	One or more occurrences of the regular expression `r`.
`r?`	Zero or one occurrence of the regular expression `r`.
`(r)`	The regular expression `r`. Used to separate a regular expression from another.
`r | r'`	The regular expression `r` or `r'`.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

2.9.3 Argument filter
Continued

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.

© Copyright 2015-2022 ABB. All rights reserved.

2.9.4 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.

© Copyright 2015-2022 ABB. All rights reserved.

2.10 FlexPendant applications

Introduction

Customized FlexPendant applications can be included in the RobotWare Add-In, simply by adding the compiled application assemblies and other resources to the file structure. No additional configuration is needed.

For the application to be loaded properly the Add-In must also register itself using the `register -type option` command in the *install.cmd* file, see *register on page 28*.

> **Note**
>
> This section only describes the deployment of FlexPendant applications. For information on creating FlexPendant applications, see the corresponding manual for the software used to create the FlexPendant application.

File structure

When adding a FlexPendant application to the Add-In, the application assemblies (*.dll*) and other resources (*.jpg, .gif, .bmp*) need to be placed in the *tps* folder in the Add-In file structure, see *File structure on page 18*.

Localized FlexPendant application

If the FlexPendant application is localized, i.e. has support for multiple languages, the language specific resources should be placed in *tps* folders under each language code folder.

For example, english resource files should be placed in the folder \<*option name*>*language**en**tps*\.

© Copyright 2015-2022 ABB. All rights reserved.

3 RobotWare Add-In Packaging tool

3.1 Introduction

3.1.1 About the RobotWare Add-In Packaging tool

General

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

The tool helps you to:

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

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

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



Tip

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

Open and licensed add-ins

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

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

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

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Installation procedure

Before installing the software make sure that the certificates are available, for more information see *Digital signing on page 78*.

	Action
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.

© Copyright 2015-2022 ABB. All rights reserved.

3.1.2 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 5, all options and additional options were system options. To be able to have one configuration of equipment for one robot and another configuration of equipment for another robot in a MultiMove system, it was necessary to make special arrangements in the *relkey.opt* file.

In RobotWare 6 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

© Copyright 2015-2022 ABB. All rights reserved.

3.1.2 Optional features
Continued

For example. PROFIenergy requires that either PROFINET Controller/Device or PROFINET Device is selected for installation:

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


© Copyright 2015-2022 ABB. All rights reserved.

3.1.3 Files of a packaged add-in

The product manifest file

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

Product details:

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

Option details:

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

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

The robot package file

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

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

This package will be transferred to the controller during installation and will be unpacked on the controller where the *.cmd* files of the add-in will be executed to install the add-in on the controller.

© Copyright 2015-2022 ABB. All rights reserved.

3.1.4 Signing with digital certificates

Digital signing

RobotWare 6 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/timstamp.dll*. (Note that it is not possible to browse to this URL.)

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

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.

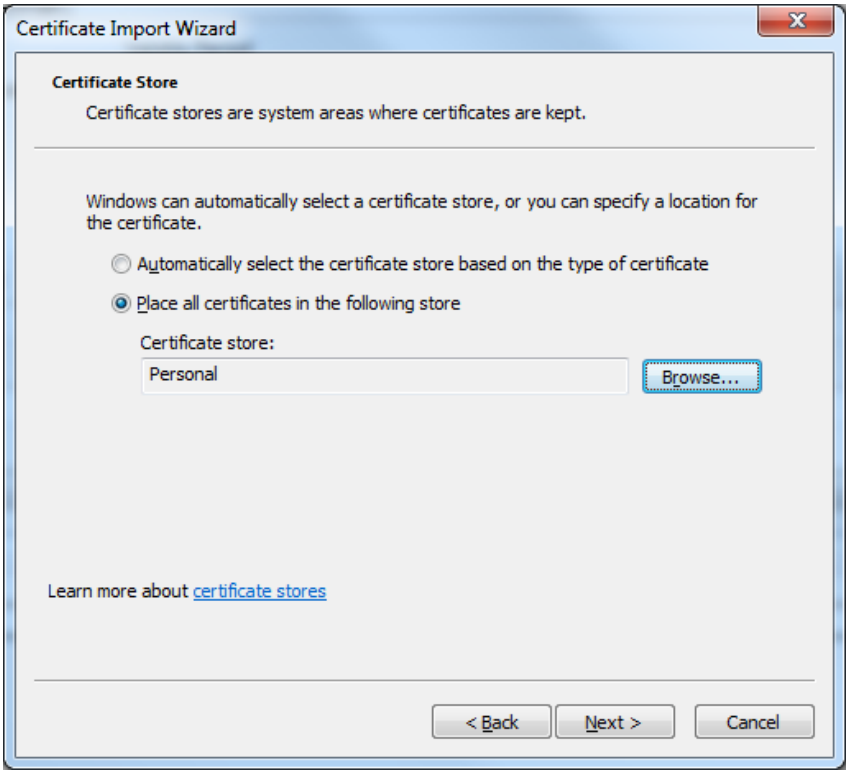
To install the certificates locate the *.pfx* file in Windows Explorer, right click on the file and select the **Install PFX** option, this will open up the **Certificate Import Wizard**.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

3.1.4 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

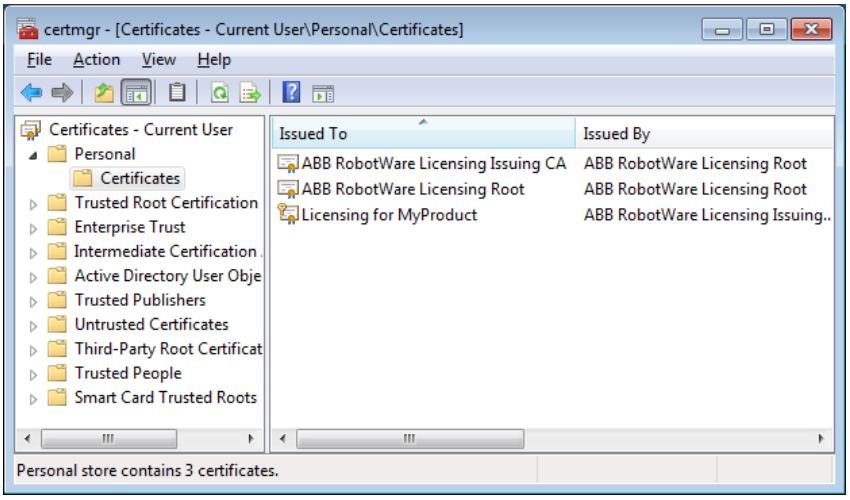
By default the wizard will try to determine the appropriate store based on the type of certificate. This would cause parts of the certificate chain to be installed as a trusted root certificate which is not recommended in the case of licensing certificates for security reasons. Instead it is recommended 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

© Copyright 2015-2022 ABB. All rights reserved.

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 *cermgr.msc* in the Windows system folder, usually *C:\Windows\system32\certmgr.msc*.



xx1500000936

© Copyright 2015-2022 ABB. All rights reserved.

3.1.5 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 83* and *Building an add-in from the console on page 99*.

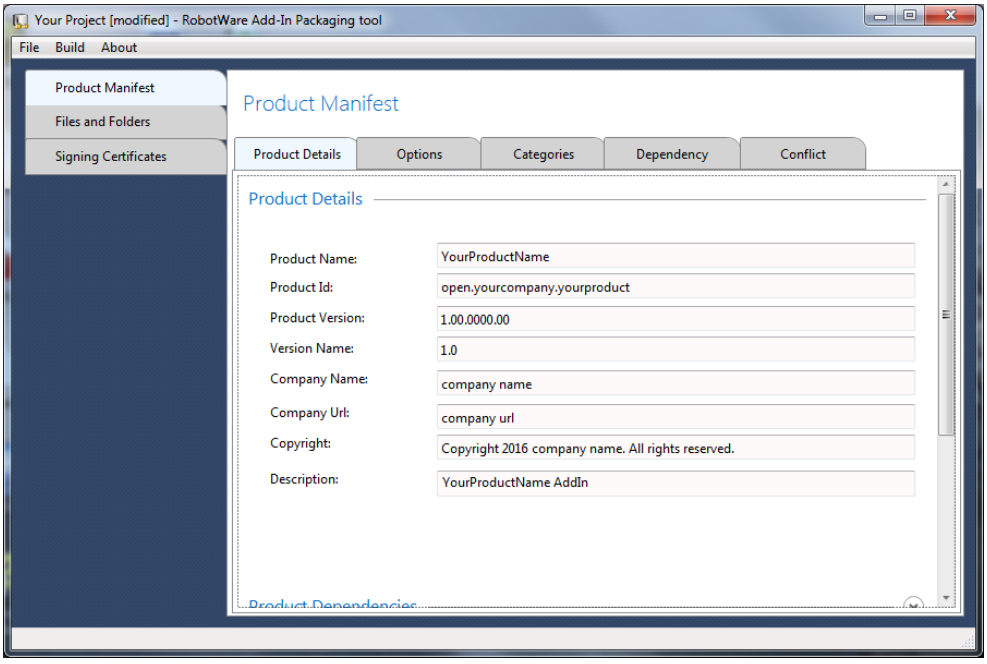
© Copyright 2015-2022 ABB. All rights reserved.

3.2 User interface

3.2.1 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.



xx1400002260

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

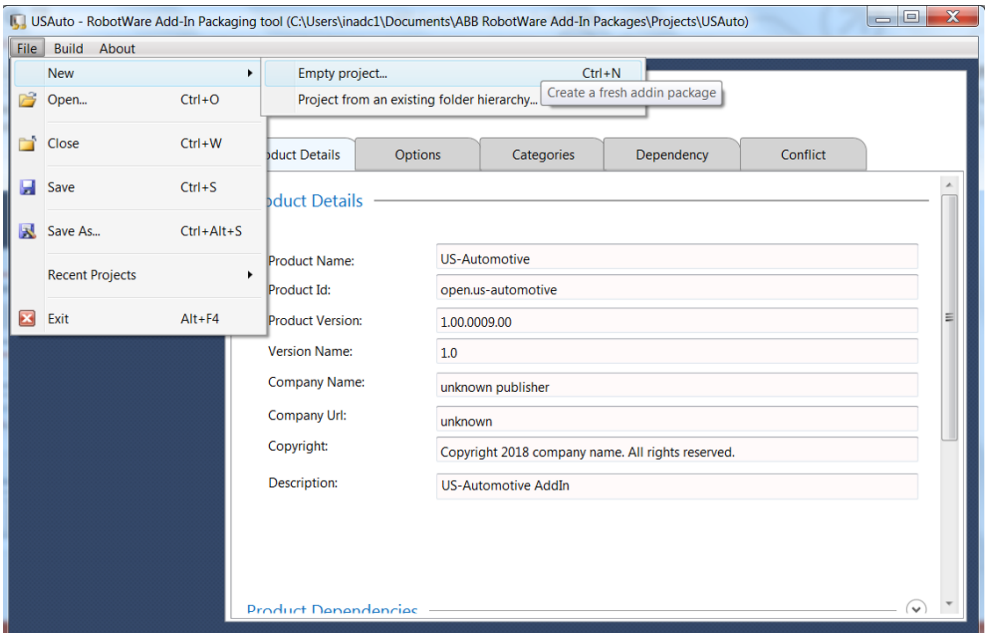
When all the information about the add-in has been entered, the add-in is built by selecting **Build Project** from the **Build** menu.

© Copyright 2015-2022 ABB. All rights reserved.

3.2.2 The File menu

The File menu

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



xx1800001606

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

Name	Description
New	Creates a new add-in project. The following options are available: • **Empty project**: Select this option to create an add-in package project from scratch. • **Project from an existing folder hierarchy**: Select this option to create an add-in using an existing 5.xx additional options. For details about creating a project, see *Creating and building an add-in project on page 97*. For details about converting an additional option to an add-in, see *Converting an additional option to an add-in on page 98*.
Open	Opens an existing add-in project. **Note** The add-in project file extension is **.rpkproj**
Close	Closes the current active project.
Save	Saves the current active project.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

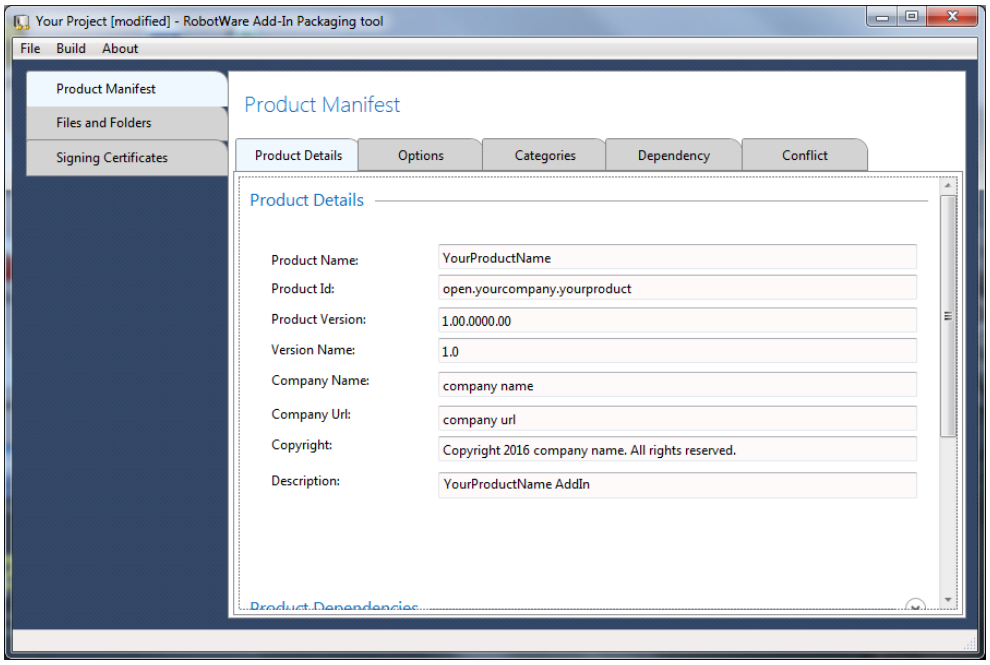
Name	Description
Save As	Save the current active project to a different location on the file system/network.  **Note** The **Save As** operation saves all the project details in project files (.rpkproj, .rpkspecs and .manifest) into the newly selected location. Also a copy of source files under the **Files and folders** tab will be created and stored under the newly selected project folder.
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.

© Copyright 2015-2022 ABB. All rights reserved.

3.2.3 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.



xx1400002260

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	The internal identifier of the product. The product id is what uniquely identifies the product. For licensed products the **Product Id** must start with one of the namespace strings defined by the licensing certificate. For more information, see *Digital signing on page 78*. For unlicensed products the **Product Id** must start with the string *open*, for example: *open.yourcompany.yourproduct*.
Product Version	The product version field is used to uniquely identify a specific build of the product. This information is used by the Installation Manager and other tools to determine the relation between different releases of a product, that is, older, equal, or newer. The format of the product version can have maximum four fields: *<major version>.<minor version>.<build version>.<revision>*

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Field name	Description
Version Name	The version name field represents the product version as displayed to the end user. It differs from the **Product Version** field in that it is intended for display purposes only and is not restricted to a specific format. It can, for example, contain identifiers such as "*Beta*" or "*Release Candidate*" in addition to the version. As a comparison RobotWare 6.02.01 has a **Product Version** of "*6.02.1029.01*" and a **Version Name** of "*6.02.01.00*". **Note** Add-Ins built with version 1.3 or older of the RobotWare Add-In Packaging tool are displayed in the Installation Manager as a combination of the **Product Name** and **Product Version** fields. From version 1.4 the **Version Name** is used instead of **Product Version** and it is therefore important that this field contains relevant information.
Company Name	The name of your company.
Company Url	The website of your company.
Copyright	Copyright information.
Description	Short product description.

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

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

Field name	Description
Identity	The internal identifier of the product.
Name	The name of the product.
Platform	The installation target platform, for instance robot controller and/or virtual controller.
Publisher	The company name of the add-in publisher.
MinVersion	The minimum product version.
MaxVersion	The maximum product version (optional).
Type	Product type. Always set to *Add-In*.

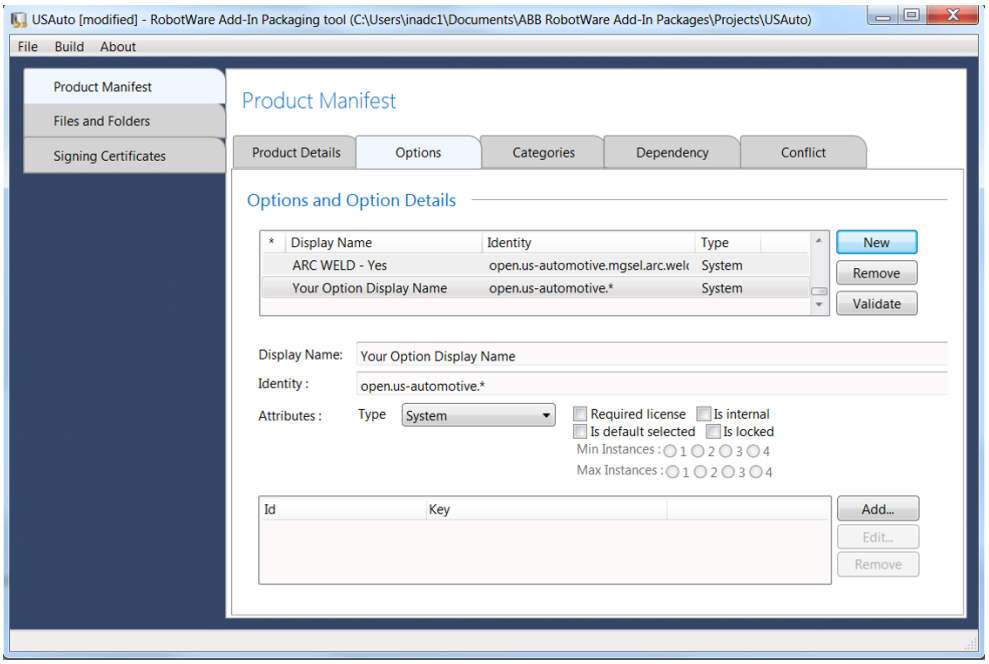
Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

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.



xx1800001609

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 Installation Manager 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 75*.
Type	Select the type of the option: • **System** - Options that are global for the system. • **Robot** - Options that can be set per robot in the system. For example, when using MultiMove where different robots may have different equipment.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

Field name	Description
Attributes	Select the option attributes: • **Required license** - The option requires a license. • **Is internal** - The option is not shown in the Installation Manager GUI. • **Is default selected** - The option is selected by default in the Installation Manager in RobotStudio. • **Is locked** - The option cannot be deselected in Installation Manager in RobotStudio. **Note** For licensed products, at least one option should have the **Required license** check box selected.
Min Instances	The minimum number of robots in the system that can have the option. **Note** This field is only valid for the option type *Robot*.
Max Instances	The maximum number of robots in the system that can have the option. For example, when using several robots in a MultiMove system. **Note** This field is only valid for the option type *Robot*.

Validate the option

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

The following validation is performed:

- The option identity must always begin with product identity text as prefix.

Feature Data

For each option it is possible to define key values that can be retrieved from *install.cmd* file during product installation. For more information see, *getkey on page 25*.

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

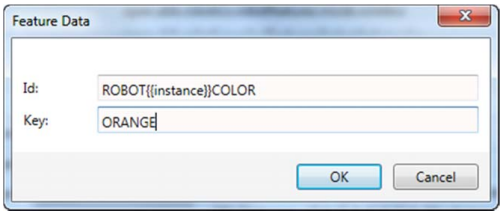
Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

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`.



xx1400002532

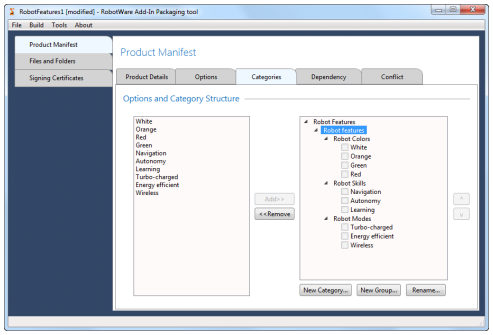
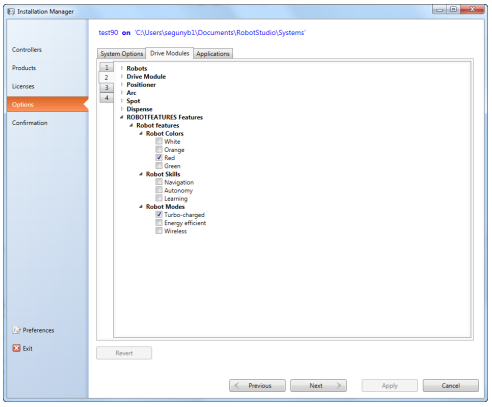
During installation, the Installation Manager will resolve `{{instance}}` to 1, 2, 3, or 4, depending on which robot this setting was meant for. This will allow to check for settings like `ROBOT1COLOR`, `ROBOT2COLOR`, `ROBOT3COLOR`, and `ROBOT4COLOR` in the *install.cmd* files, for example in the following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

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

Categories tab

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

It is not allowed to mix system options and robot options within the same category. When both system options and robot options are included in the add-in, they must be put into separate categories.

Categories tab	Installation Manager
 xx1400002382	 xx1400002383

Continues on next page

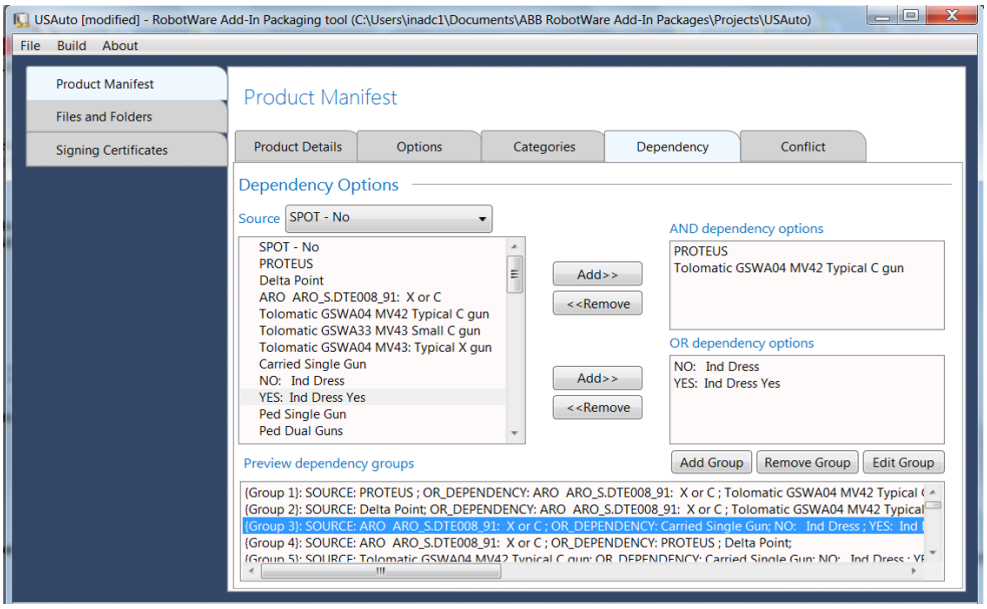
© Copyright 2015-2022 ABB. All rights reserved.

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.



xx1800001610

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 75*.

> **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.

Continues on next page

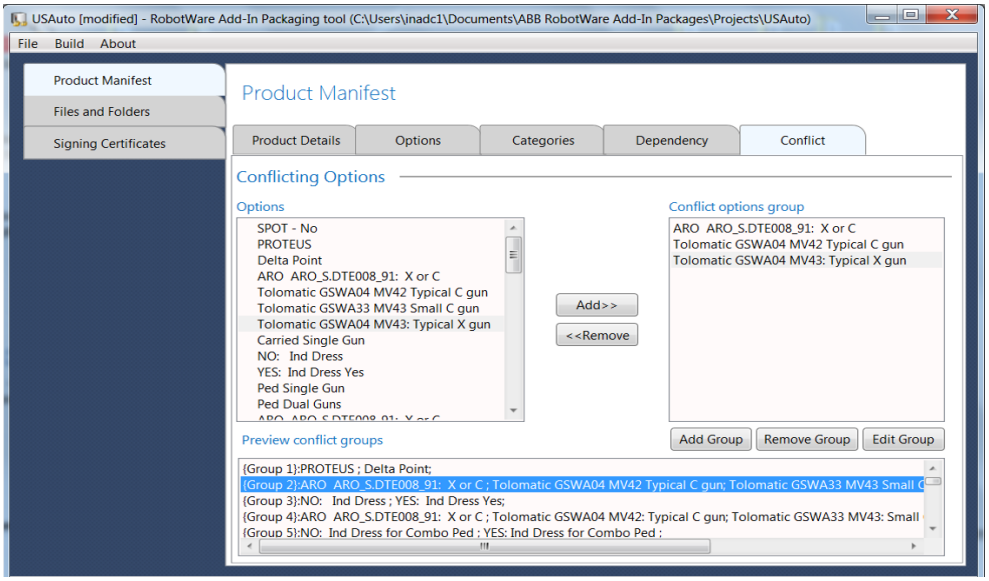
© Copyright 2015-2022 ABB. All rights reserved.

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

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

Conflict tab

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



xx1800001773

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

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

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

The Files and folders view

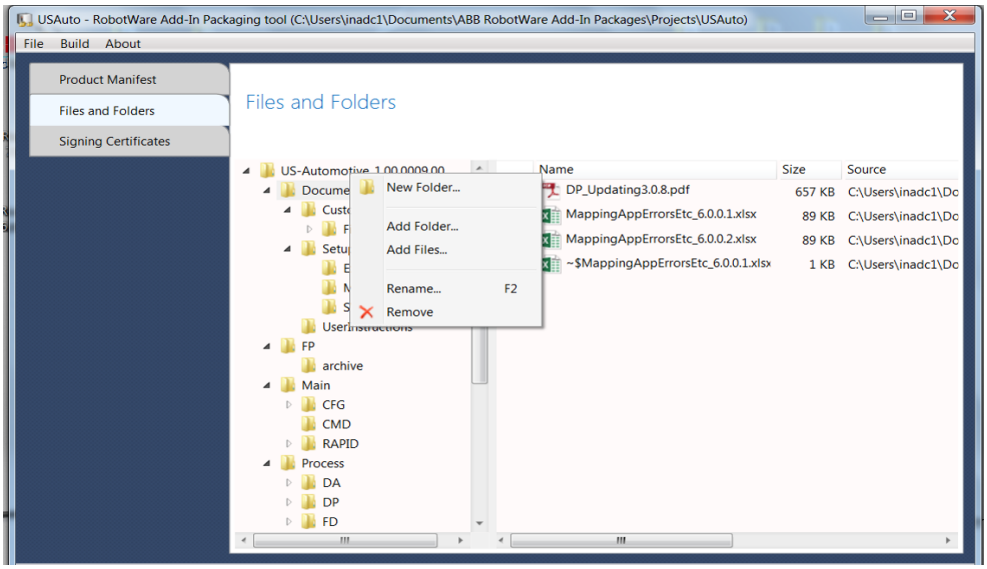
The **Files and Folders** view is used to create the robot package file.

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

© Copyright 2015-2022 ABB. All rights reserved.

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

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



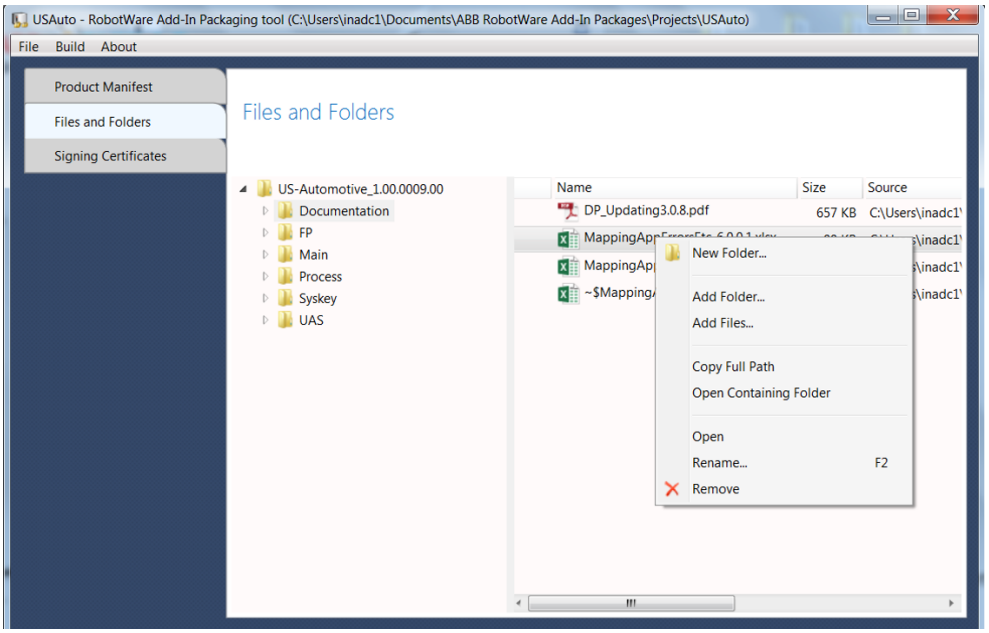
xx1800001774

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	Renames the selected folder. **Note** The root folder cannot be renamed.
Remove	Removes the selected folder. **Note** The root folder cannot be removed.

© Copyright 2015-2022 ABB. All rights reserved.

3.2.3 The Product Manifest view
Continued

Right-click inside a folder for the following options:



xx1800001775

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.

© Copyright 2015-2022 ABB. All rights reserved.

- 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.

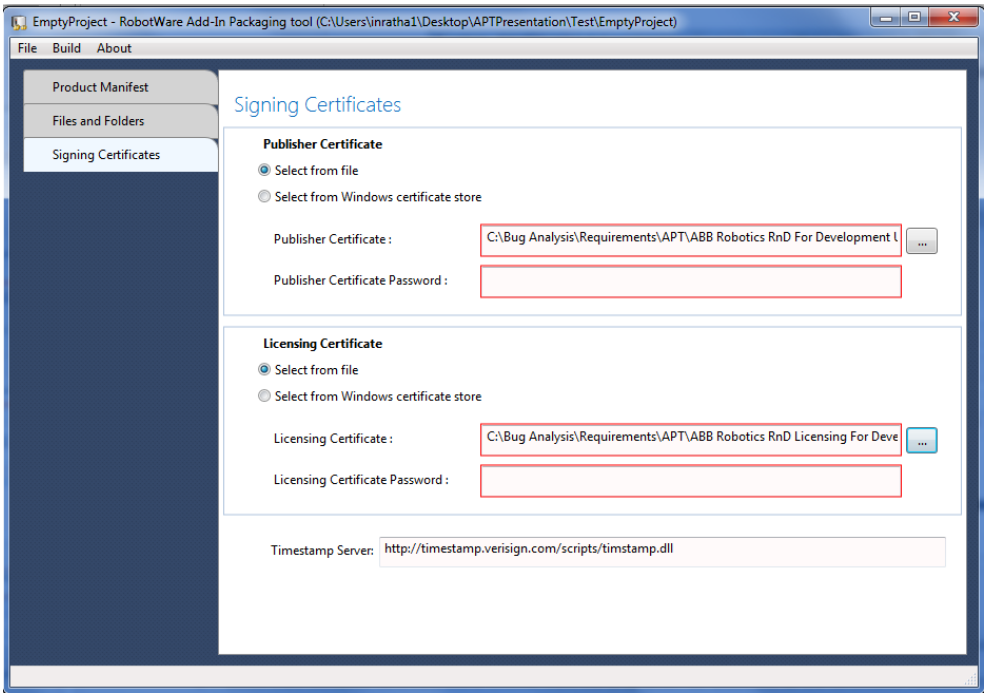
Files and folders for converted add-ins

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 78*.



xx1800001776

© Copyright 2015-2022 ABB. All rights reserved.

3.2.3 The Product Manifest view
Continued

Section	Field name	Description
Publisher Certificate	**Select from file**	Select this option if the publisher certificate for digital signing should be provided as *.pfx* files. **Note** Browse **Publisher Certificate** to select the certificate from its stored location.
	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	Browse to select a certificate (*.pfx*file) from its stored location. The selected path is displayed. **Note** This field is used in combination with option **Select from file**.
	Publisher Certificate Password	The password for the publisher certificate when specified as a *.pfx* file.
Licensing Certificate	**Select from file**	Select this option if the licensing certificate for digital signing should be provided as *.pfx* files. **Note** Browse **Licensing Certificate** to select the certificate from its stored location.
	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	Browse to select a certificate (*.pfx*file) from its stored location. The selected path is displayed. **Note** This field is used in combination with option **Select from file**.
	Licensing Certificate Password	The password for the licensing certificate when specified as a *.pfx* file.
	Timestamp Server	Displays the URL to a timestamp server. For more information, see *Timestamping on page 78*.

© Copyright 2015-2022 ABB. All rights reserved.

3.3 Creating and building an add-in project

Procedure

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

Step	Action	Description
1	Create a new empty project by clicking **New** and then **Project** from the **File** menu. Or: Select **Project from an existing folder hierarchy**, then the tool will try to generate default data for the add-in.	
2	Enter all the information about the add-in in the tabs of the product manifest view.	*The Product Manifest view on page 86*
3	Add all files and folders.	*The Files and folders view on page 92*
4	For licensed options, add the publisher and licensing certificates.	*The Signing Certificate view on page 95*
5	Build the add-in by selecting **Build Project** from the **Build** menu.	
6	Generate a license using the License Generator.	*License Generator on page 101*
7	Verify the add-in by building a system using the Installation Manager in RobotStudio.	*Operating manual - RobotStudio*

© Copyright 2015-2022 ABB. All rights reserved.

3.4 Converting an additional option to an add-in

Overview

This section describes how to package an existing RobotWare 5 additional option as a RobotWare 6 add-in.

The RobotWare add-in Packaging tool will add information from the imported additional option as a template add-in. It is necessary to go through all the option details one by one to verify that the content is correct.

Limitations

Following are the limitations of RobotWare Add-In Packaging tool while converting an additional option to an add-in:

- The RobotWare Add-In Packaging tool does not perform any migration of the additional option to RobotWare 6, it is only a packaging tool. It is mandatory to first migrate the additional option to RobotWare 6, using the migration tool in RobotStudio.
- If the additional option has encrypted *relkey.opt* and *.dat* files, the files needs to be decrypted. The RobotWare Add-In Packaging tool will not be able to process any encrypted files.

Procedure

Use the following procedure to convert an additional option to an add-in.

	Action	See
1	Convert the project by clicking **Project from an existing folder hierarchy** from the **File** menu.	
2	Verify the information about the add-in in the tabs of the **Product Manifest** view.	*The Product Manifest view on page 86*
3	Verify that all files and folders are present in the **Files and folders** view.	*The Files and folders view on page 92*
4	For licensed options, add the publisher and licensing certificates.	*The Signing Certificate view on page 95*
5	Build the add-In by selecting **Build Project** from the **Build** menu.	
6	Generate a license using the License Generator.	*License Generator on page 101*
7	Verify the add-in by building a system using the Installation Manager in RobotStudio.	*Operating manual - RobotStudio*

© Copyright 2015-2022 ABB. All rights reserved.

3.5 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
`-projectFileName`	Project file name for APT `RPKProj` file.
`-pubcertfile`	Publisher signing certificate file.
`-pubcertfilepassword`	Password for the publisher certificate.
`-liccertfile`	Licensing signing certificate file.
`-liccertfilepassword`	Password for the licensing certificate
`-liccertthumbprint`	Thumbprint for the licensing certificate stored in the certificate store.
`-pubcertthumbprint`	Thumbprint for the publisher certificate stored in the certificate store.
`-timestampurl`	Timestamping server URL for code signing.
`-outputfolder`	Output folder where project output will be generated.
`-isopenaddin`	If this parameter's value is set to TRUE, an open add-in is generated without considering the licensing.

Continues on next page

© Copyright 2015-2022 ABB. All rights reserved.

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.

© Copyright 2015-2022 ABB. All rights reserved.

4 License Generator

4.1 Introduction

General

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

Installation procedure

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

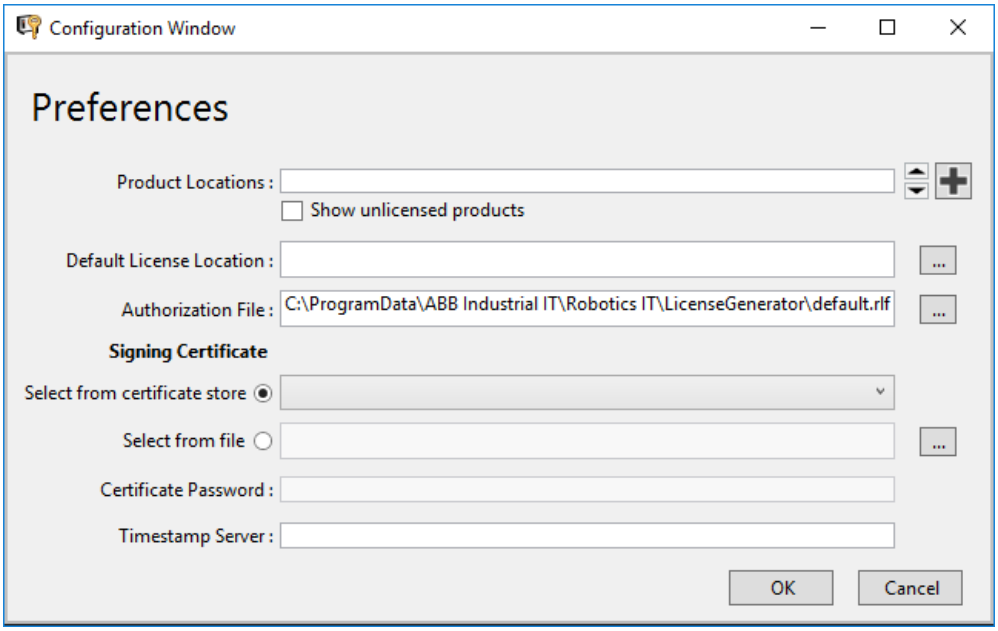
© Copyright 2015-2022 ABB. All rights reserved.

4.2 The user interface

4.2.1 The Preferences window

Preferences

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



xx2000002041

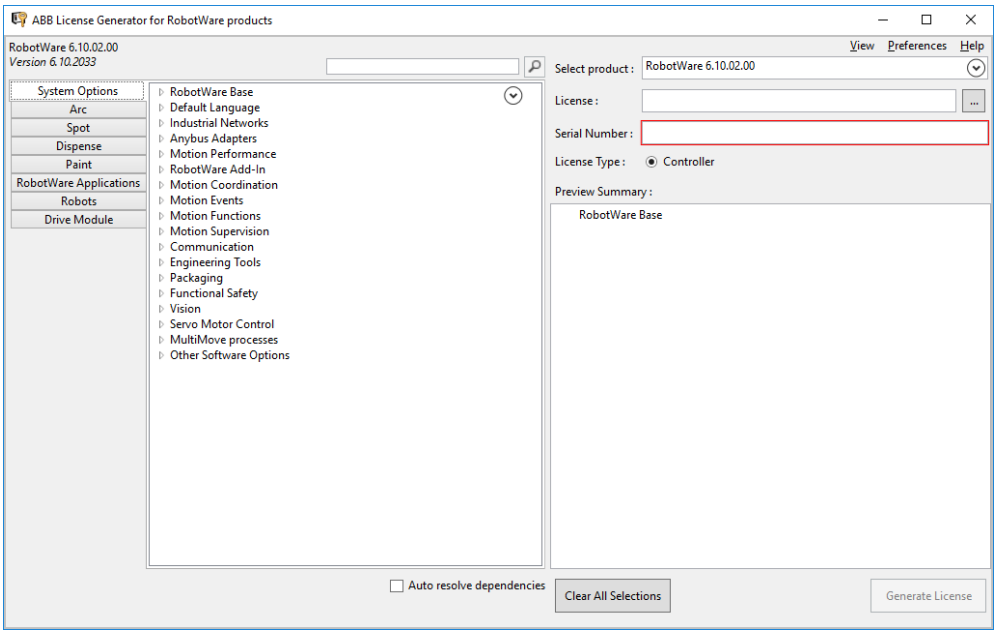
Field name	Description
Product Locations	The location of the product manifest files (**.rmf*).
Default License Location	The default location where the created licence files (**.rlf*) should be saved.
Authorization File	The authorization file, license file, (**.rlf*) for the License Generator provided by ABB.
Signing Certificate (radio button)	Install/use the certificate provided by ABB, same certificate as for the RobotWare Add-In Packaging tool. • **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 78*.

© Copyright 2015-2022 ABB. All rights reserved.

4.2.2 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

© Copyright 2015-2022 ABB. All rights reserved.

4.2.2 The main window
Continued



Tip

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

© Copyright 2015-2022 ABB. All rights reserved.

4.3 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 102*.

Use this procedure to create a new license.

	Action
1	Select a product manifest to create the license for in the **Select Product** field.
2	Type in the serial number of the controller.
3	Select all options in the tree-view.
4	Generate the license file by clicking the **Generate License** button.
5	Verify the license by building a system using the Installation Manager in RobotStudio.

Import and modify a license

Before creating a license it is necessary to set up the preferences in the **Preferences** window, see *The Preferences window on page 102*.

Use this procedure to import and modify a license.

	Action
1	Select the product manifest you want to create the license for in the **Select Product** field.
2	Select the license to import in the **Order/License** field. Click **Open** to import the license.
3	Type in the serial number of the controller.
4	(Optional) Add or remove options in the tree-view.
5	Generate the license file by clicking the **Generate License** button.
6	Verify the license by building a system using the Installation Manager.

Viewing a licence file

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

Use this procedure to view a license.

	Action
1	Click **View** on the main menu.
2	Browse to the folder where the license is located.
3	Select the license file and click **Open**.
4	The content of the license file is displayed.

© Copyright 2015-2022 ABB. All rights reserved.

This page is intentionally left blank

Index

$, 22
#, 22

C
CAB_TASK_MODULES, 43
CIRCLEMOVE, 15
config, 22
copy, 24

D
default argument values, 48
delay, 24
delete, 24
direxist, 25

E
echo, 25
eio.cfg, 47
event log messages, 35
event log texts, 36
event log titles, 38
example, 15, 19

F
fileexist, 25
file structure, 17
find_replace, 25

G
getkey, 25
goto, 26

I
ifstr, 26
ifvc, 26
include, 26
install.cmd, 22, 32

L
load modules, 43

M
math_lib_set_mem_size, 27
mkdir, 27
mmc.cfg, 48
module, 33
MoveCircle, 33

N
NOSTEPIN, 33

O
onerror, 27

P
pick list, 48
print, 28

R
rapid_delete_palette, 28
rapid_delete_palette_instruction, 28
RAPID module, 33
RAPID rules, 48
register, 28

S
safety, 11
selections, 14
setenv, 29
setstr, 30
sys.cfg, 43
system module, 33

T
template files, 35
text, 30
timestamp, 31

V
validating .xml, 39
version.xml, 19

X
xml
  validating, 39

© Copyright 2015-2022 ABB. All rights reserved.

ABB AB
Robotics & Discrete Automation
S-721 68 VÄSTERÅS, Sweden
Telephone +46 (0) 21 344 400

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

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

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

abb.com/robotics

3HAC051193-001, Rev K, en

© Copyright 2015-2022 ABB. All rights reserved.
Specifications subject to change without notice.