Application manual

PC SDK

5.13

Document ID: 3HAC036957-001

Revision: -

© Copyright 2010 ABB. All rights reserved.

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, damages 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, and contents thereof must not be imparted to a third party nor be used for any unauthorized purpose. Contravention will be prosecuted.

Additional copies of this manual may be obtained from ABB at its then current charge.

© Copyright 2010 ABB All rights reserved.

ABB AB
Robotics Products
SE-721 68 Västerås
Sweden

© Copyright 2010 ABB. All rights reserved.

Overview . . . 7
Product documentation, M2004 . . . 9
Safety . . . 11

1 Introduction 13

1.1 About creating controller applications . . . 13
1.2 Documentation and help . . . 15
1.3 Terminology . . . 17

2 Installation and development environment 19

2.1 Installation overview . . . 19
2.2 How to obtain and install a license key for RAB 5.09 or earlier . . . 22
2.3 How to set up your PC to communicate with robot . . . 23
2.4 Development environment . . . 25
2.5 Two development models - virtual and real . . . 27
2.6 Conversion of VS 2005 projects to Visual Studio 2008 . . . 29

3 Run-time environment 31

3.1 Overview . . . 31
3.2 Running PC SDK Applications . . . 32
3.2.1 Overview . . . 32
3.2.2 Mastership . . . 33
3.2.3 PC application configuration . . . 35
3.2.4 Communication between PC and controller . . . 38
3.2.5 Licence verification - applies only to versions earlier than PC SDK 5.10 . . . 39
3.3 Release upgrades and compatibility . . . 40

4 Developing Controller applications 41

4.1 Introduction . . . 41
4.2 Analysis and design . . . 42
4.3 Controller events and threads . . . 44
4.4 User Authorization System . . . 46
4.5 Exception handling . . . 48
4.6 How to use the online help . . . 50

5 Using the PC SDK 51

5.1 Controller API . . . 51
5.2 Create a simple PC SDK application . . . 53
5.3 Discovery domain . . . 61
5.4 Accessing the controller . . . 63
5.5 Rapid domain . . . 69
5.5.1 Working with RAPID data . . . 69
5.5.2 Handling arrays . . . 77
5.5.3 ReadItem and WriteItem methods . . . 80
5.5.4 UserDefined data . . . 81
5.5.5 RAPID symbol search . . . 89
5.5.6 Working with RAPID modules and programs . . . 94
5.5.7 Enable operator response to RAPID UI-instructions from a PC . . . 97
5.6 IO system domain . . . 102
5.7 Event log domain . . . 109
5.8 Motion domain . . . 111
5.9 File system domain . . . 113
5.10 Messaging domain . . . 116

© Copyright 2010 ABB. All rights reserved.

6 PC - Debugging and troubleshooting **127**

6.1 Debugging 127
6.2 Troubleshooting 131

7 Deployment of a PC SDK application **133**

7.1 Overview 133

© Copyright 2010 ABB. All rights reserved.

© Copyright 2010 ABB. All rights reserved.

© Copyright 2010 ABB. All rights reserved.

Overview

About this manual

ABB's PC Software Development Kit (PC SDK) is a software tool, which enables programmers to develop customized operator interfaces for the IRC5 robot controller.

The purpose of this manual is to help software developers get started with PC SDK application development.

NOTE!

Earlier till version 5.12 PC SDK was a part of RAB (Robot Application Builder).

Usage

PC SDK application manual covers application development using PC SDK. Code samples are written in C# and Visual Basic. Please note that there exists a separate manual *(Application Manual - FlexPendant SDK)* for FlexPendant application development.

Who should read this manual?

This manual is mainly intended for software developers, who use PC SDK to create robot applications adapted to end-user needs, and RobotStudio add-ins that communicates with virtual or real controller. It is also useful for anyone who needs an overview of doing controller applications.

Prerequisites

The reader should

- be familiar with IRC5, the FlexPendant, and RobotStudio.
- be used to Microsoft Visual Studio and Windows programming.
- be familiar with one of the .NET programming languages C# or Visual Basic.NET.
- be used to object oriented programming.

Organization of chapters

The manual is organized as follows:

Chapter	Contents
1.	Introduction. Terminology. Safety.
2.	Installation and setup. Development environment . Virtual robot technology.
3.	Software architecture. Run-time environment for PC applications. How clients access controller resources and communicate with the robot controller. Application configuration. Upgrades and compatibility.
4.	Developing PC SDK applications. Analysis and design. Important programming issues: controller events and threads, UAS, exception handling. Online help.
5.	Using the PC SDK. How to add controller functionality using the Controller API. Programming issues and code samples in VB and C#.
6.	Testing, debugging and troubleshooting PC SDK applications. Using printouts, error codes in exceptions and so on. Checklist for contacting a service organization.

Continues on next page

© Copyright 2010 ABB. All rights reserved.

Continued

References

Reference	Document ID
Operating manual - IRC5 with FlexPendant	3HAC 16590-1
Operating manual - RobotStudio	3HAC032104-001
Operating manual - Getting started, IRC5 and RobotStudio	3HAC027097-001
Technical reference manual - RAPID Instructions, Functions and Data types	3HAC16581-1
Technical reference manual - System parameters	3HAC17076-1
Application manual - FlexPendant SDK	3HAC036958-001
Application manual - Robot communication and I/O Control.	3HAC020435-001

Revisions

Revision	Description
-	First edition From PC SDK 5.13 onwards this manual replaces: *Application manual - Robot Application Builder (3HAC028083-001)* For information on FlexPendant SDK refer *Application manual - FlexPendant SDK*

© Copyright 2010 ABB. All rights reserved.

Product documentation, M2004

General

The robot documentation may be divided into a number of categories. This listing is based on the type of information contained within the documents, regardless of whether the products are standard or optional. This means that any given delivery of robot products will not contain all documents listed, only the ones pertaining to the equipment delivered. However, all documents listed may be ordered from ABB. The documents listed are valid for M2004 robot systems.

However, all documents listed may be ordered from ABB. The documents listed are valid for M2004 robot systems.

Product manuals

All hardware, robots and controllers, are delivered with a Product manual, which is divided into two parts:

Product manual, procedures

- Safety information
- Installation and commissioning (descriptions of mechanical installation, electrical connections)
- Maintenance (descriptions of all required preventive maintenance procedures including intervals)
- Repair (descriptions of all recommended repair procedures including spare parts)
- Additional procedures, if any (calibration, decommissioning)

Product manual, reference information

- Safety information
- Reference information (article numbers for documentation referred to in Product manual, procedures, lists of tools, safety standards)
- Part list
- Foldouts or exploded views
- Circuit diagrams

The product manual published as a PDF consists of only one file where the two parts are presented together, as one Product manual.

Technical reference manuals

The following manuals describe the robot software in general and contain relevant reference information:

Product manual, procedures

- **RAPID overview**: An overview of the RAPID programming language.
- **RAPID Instructions, Functions and Data types**: Description and syntax for all RAPID instructions, functions and data types.
- **System parameters**: Description of system parameters and configuration workflow

© Copyright 2010 ABB. All rights reserved.

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, CD with PC software)
- How to use the application
- Examples of how to use the application

Operating manuals

This group of manuals is aimed at those having first hand operational contact with the robot, that is, production cell operators, programmers and troubleshooters. It includes:

- **Getting started - IRC5 and RobotStudio**
- **IRC5 with FlexPendant**
- **RobotStudio**
- **Troubleshooting**

© Copyright 2010 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 of regulations

Before beginning work with the robot, make sure you are familiar with the safety regulations described in *Operating manual - IRC5 with FlexPendant.*

© Copyright 2010 ABB. All rights reserved.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

1 Introduction

1.1. About creating controller applications

Flexible user interfaces

Robots are usually delivered with a general operator interface. However, different processes require different operator handling and customers need flexible solutions where the user interface is adapted to user specific needs.

PC SDK allows system integrators, third parties or end-users to add their own customized operator interfaces for the IRC5 controller. Such custom applications can be realized as independent PC applications, which communicate with the robot controller over a network. You can also use PC SDK to communicate with a virtual or real controller from a RobotStudio add-in.

For FlexPendant based applications please refer the FlexPendant SDK application Manual.

NOTE!

Controller applications are not platform independent. You must choose to develop the application for either the PC platform or the FlexPendant (refer FlexPendant SDK application Manual).

Local vs Remote client

The difference between the two platforms is that a PC application is a remote client, whereas a FlexPendant application is a local client.

Remote clients do not have all the privileges of a local client. For example, both PC and FlexPendant applications can reset the program pointer and start RAPID execution, for example, but for a PC SDK application to do this there are certain restrictions. Mastership of the Rapid domain must be requested explicitly by the application programmer and the IRC5 controller has to be in automatic operating mode.

An advantage of a remote client, on the other hand, is the possibility to monitor and access several robot controllers from one location. As for large applications the PC platform is also less limited than the FlexPendant as regards memory resources and process power.

NOTE!

A minimum response time for a real controller should be expected to be in the order of 10-100 milliseconds, meaning that hard real time demands cannot be met on any platform. For more information, see *Communication between PC and controller on page 38*.

Ease-of-use on the factory floor

A well-designed user interface presents relevant information and functionality at the right time. In this respect, customized user interfaces are clearly very desirable to the end-user. As tailored solutions are easier to operate, they also optimize user's investment in automation.

PC SDK enables customized user interfaces for IRC5. It is important to keep in mind, however, that PC SDK itself does not guarantee increased customer value. To achieve this, PC SDK applications should be developed with care and with a heavy emphasis placed on ease-of-use. Understanding end-users' needs is in fact crucial to realizing the benefits of customized interfaces.

Continues on next page

© Copyright 2010 ABB. All rights reserved.

Continued

.NET and Visual Studio

PC SDK uses Microsoft .NET and Microsoft Visual Studio. It is thus assumed that you know how to program Windows platforms using Visual Studio. Among programmers .NET distinguishes itself by the programming model provided by the Microsoft .NET Framework.

One feature is the programming language independence, leaving the choice to the developer to use any language provided by the integrated development environment Visual Studio. In PC applications any of the .NET languages should work, but ABB support is only offered for Visual Basic and C#.

For a Windows programmer familiar with Visual Studio and .NET, developing a customized operator view is rather straight-forward.

Considerable efforts have been made to allow controller application developers to start working without having to overcome a steep learning curve. To further speed up the development process, the virtual IRC5 of RobotStudio can be used to test and debug controller applications.



NOTE!

Some knowledge in Windows programming, object orientation and a .NET programming language is necessary to use PC SDK

Robustness and performance

Developing an application using PC SDK involves isues related to performance and reliability that one needs to know about before getting started.

It is strongly advisable to read this manual to learn about specific PC SDK issues while moving to PC SDK development.



NOTE!

Please read this manual and the release notes before starting to code.

© Copyright 2010 ABB. All rights reserved.

1.2. Documentation and help

Introduction

PC SDK includes an extensive on-line help module, which comes with the installation of the SDK. After having installed RobotStudio, by clicking Windows' **Start** menu, then pointing at Programs > ABB Industrial IT > Robotics IT > RobotStudio 5.xx > SDK you will find:

- *Application manual PC SDK*
- *Reference Manual PC SDK*

Application manual

This *Application manual PC SDK*, is the recommended way to get started if you are new to PC SDK development. It explains how PC SDK works. It has code examples in C# and VB.NET and provides hands-on exercises.

The application Manual is provided in two formats, HTML Help and PDF. HTML is the recommended format for the PC screen and PDF is the best choice if you want printouts.

NOTE!

The Application Manuals PDF can be found in the installation directory, at C:\Program Files\ABB Industrial IT\Robotics IT\SDK\.

SDK Reference Help

The PC SDK Reference Help files should be used while programming.

It makes up the complete reference to the class libraries in PC SDK. Method signatures are provided in C# and Visual Basic.

Please note that they are **not** integrated with the Visual Studio Help function. Clicking F1 when pointing at code, for example, will open the *Visual Studio Programmer's Reference* or the *.NET Framework Class Library* for the specific language and topic. Many times this is what you want, but if your problem is PC SDK related you need to open the appropriate SDK Reference Help to find a solution.

NOTE!

You are recommended to keep the help files open while programming, as you will frequently need them for PC SDK related issues.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

RobotStudio Community

ABB Robotics launched a community named *RobotStudio Community*, for its PC Software users. The *Developer Tools* in *Developer Section* of *RobotStudio Community* has information and some videos about programming with the PC SDK. At *Content Sharing* there is a complete FlexPendant SDK application available for download. It is recommended for average users and for beginners.

ABB encourage open conversations and believe everyone has something to contribute. The *User Forum* of *RobotStudio Community* has a section dedicated to Robot Application development. Here beginners as well as experts discuss code and solutions online. If you are facing a coding problem the *User Forum* should be your first choice, as there is a good chance that someone will give you the help you need to proceed.

RobotStudio Community also provides the means to share code and videos. Your contribution will be appreciated. Working together is many times the key to success.



TIP!

Try it out at *www.abb.com/robotics > RobotStudio Community*.

MSDN

MSDN (Microsoft Developer Network) at www.msdn.com is a one of many sources of information for general programming issues related to .NET and Visual Studio.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

1.3. Terminology

About terms and acronyms

Some terms used in this manual are product specific and crucial for understanding. Moreover, acronyms, words formed from initial letters, are sometimes used instead of long terms. To avoid confusion, important terminology is clarified below.

Definitions

Term	Definition
C# and Visual Basic.NET	.NET programming languages.
Common Language Runtime	The core runtime engine in the .NET Framework for execution of managed code. Provides services such as cross-language integration, code access security, object lifetime management, and debugging and profiling support.
Controller Application Programming Interface	The public class libraries of PC SDK, which offer robot controller functionality. Also referred to as CAPI.
Device	The FlexPendant is a “smart device” in the .NET vocabulary, that is, a complete computer in itself with its own processor, operating system and so on.
FlexPendant	ABB’s hand held device, used with IRC5 robot controller. It is developed with Microsoft’s technology for embedded systems, Windows CE and .NET Compact Framework.
IRC5	ABB’s robot controller.
JIT compiler	When compiling managed code, the compiler translates the source code into Microsoft Intermediate Language (MSIL), which is a CPU-independent set of instructions. Before code can be executed, MSIL must be converted to CPU-specific code, usually by a just-in-time (JIT) compiler.
managed code	Code that is executed and managed by the Microsoft .NET Framework’s common language runtime. All code produced by Visual Studio executes as managed code.
Microsoft Visual Studio	The integrated development environment that developers work inside when using the .NET Framework.
Microsoft .NET Framework	An integral Windows component supporting the building and running of applications.
Network socket	A communication end-point unique to a machine communicating on an Internet Protocol-based network.
PC SDK programmer	A programmer who uses PC SDK to develop custom applications.
PC SDK application	A custom application developed with PC SDK.
Robot Application Builder	ABB software tool, which enabled the development of custom operator interfaces for IRC5. Often referred to as RAB. The RAB is split to FlexPendant SDK and PC SDK. Robot Application Builder (RAB) was a software tool, which enabled programmers to develop customized FlexPendant or PC interfaces for the IRC5 robot controller.
Robot Communication Runtime	The communication layer used by Controller API to communicate over the network with an IRC5 controller.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

Term	Definition
unmanaged code	Code that is executed directly by the operating system, outside the .NET Framework. Unmanaged code must provide its own memory management, type checking, and security support, unlike managed code, which receives these services from the common language runtime. All code executing in the robot controller, as well as part of the code executing in the FlexPendant is unmanaged.
Virtual IRC5	Virtual robot technology makes it possible to run a virtual IRC5 controller, virtual mechanical units and a virtual FlexPendant on the desktop. Included as freeware in ABB's RobotStudio.
Windows CE	The embedded operating system running on the FlexPendant device.

Acronym	Definition
CAPI	Controller Application Programming Interface
CLR	Common Language Runtime
GUI	Graphical User Interface
MSDN	Microsoft Developer Network, source of information for .NET developers at: *www.msdn.com*
PC SDK	Personal Computer Software Development Kit
SDK	Software Development Kit
TCP/IP	Transmission Control Protocol (TCP) and Internet Protocol (IP)
VB	Visual Basic
VS	Visual Studio

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

2 Installation and development environment

2.1. Installation overview

About this section

This section describes how to install PC SDK. When the installation is complete, you can program, compile and test PC applications for the IRC5 controller.

Supported platforms

The following software requirements have to be met:

- Operating system: Microsoft Windows XP + SP2, Windows Vista+ SP2 or Windows 7
- Microsoft Visual Studio 2005 Express or better, or, Microsoft Visual Studio 2008 Express or better.
- .NET Framework 2.0 SP2

The following hardware requirement have to be met:

- 50 MB free disk-space on the installation disk

NOTE!

The RobotWare system of a real IRC5 controller to connect to the PC SDK application must have the option *PC Interface*.

NOTE!

PC SDK is developed and tested for the English version of Visual Studio. If you are running Visual Studio in another language you are recommended to switch to the English version.

Requirements for installing and using PC SDK

PC SDK is installed while you install RobotStudio. For more information on installing RobotStudio see *Installing and Licensing RobotStudio* in *Operating manual - RobotStudio*. To use PC SDK, the following requirements have to be met. Also make sure that you have administrator permissions on the computer that you are using.

Before...	you must...
debugging using a virtual IRC5	learn how to run the virtual IRC5 in RobotStudio.
executing the application targeting a real IRC5 system	check that the robot system has the controller option *PC Interface* (for PC applications). Set up a connection between your PC and the robot controller. For more information, see *How to set up your PC to communicate with robot on page 23* for details about how this is done.

NOTE!

The Visual Studio installation installs .NET Framework 2.0.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

About the PC SDK installation

Starting wtih RobotStudio 5.13, both PC SDK and FlexPendant SDK is included in the RobotStudio installation. .NET assemblies are installed in the Global Assembly Cache (GAC). RobotStudio installs PC SDK 5.13, side by side with previously installed versions of PC SDK. Previously PC SDK was a part of Robot Application Builder (RAB) which also included FlexPendant SDK. RAB 5.12 was the last release of Robot Application Builder.

RAB 5.11 to 5.12

Installs PC SDK and FlexPendant SDK side by side with any previously installed versions. This makes it easier to work with several versions of the PC SDK on a single computer.

RAB 5.10

RAB 5.10 upgraded any previously installed PC SDK to 5.10 and installed FlexPendant SDK 5.08, 5.09 and 5.10 side-by-side. The reason for the side-by-side installation of several FlexPendant SDK versions was to make it easier for FlexPendant SDK users to work on FlexPendant SDK applications targeting different RobotWare versions. Earlier RAB releases can be downloaded from *http://www.abb.com/robotics* > **RobotStudioCommunity** >Developer Tools > PC SDK Overview.

What is installed?

The following are installed on your PC:

- SDK assemblies and resources
- *Application manual PC SDK*
- *Reference Manual PC SDK*

Working with several versions

A PC SDK application normally targets a specific RobotWare release. Assuming you are developing a PC SDK application for a new customer who uses RobotWare 5.12 and simultaneously you are maintaining an existing PC SDK application for a customer with robot system using RobotWare 5.09, then you will need to work with two different PC SDK releases on your PC. For details about releases and compatibility, see *Release upgrades and compatibility on page 40*.

PC applications

If you install PC SDK 5.13 and previous versions of PC SDK which came along with FlexPendant SDK as Robot Application Builder, the previous versions will exist on the PC. You need to choose which PC SDK version to use when adding PC SDK references to the application project in Visual Studio (browse to the installation directory that matches the version when adding the PC SDK references to the project). You should also set the Reference Property *Specific Version* to *true* to ensure that the correct version of the PC SDK dlls in the Global Assembly Cache (GAC) is used in run-time.

RobotStudio Add-In

A RobotStudio Add-In that uses PC SDK cannot decide which version of the PC SDK assemblies to use during runtime. The reason being an Add-In itself is an assembly that is loaded into the RobotStudio application domain. As RobotStudio also uses PC SDK

Continues on next page

© Copyright 2010 ABB. All rights reserved.

Continued

internally, the PC SDK assemblies are already loaded, and the Add-Ins are forced to use the same version, which is the same version as RobotStudio. For example, an Add-In that is loaded into RobotStudio 5.13 will be forced to use the PC SDK 5.13 assemblies.

Installation procedure

The installation procedure is very simple. By default PC SDK is installed when you install RobotStudio. For more information, see *Installing RobotStudio*, in *Operating Manual - Getting started, IRC5 and RobotStudio*.

An installation wizard will guide you through the installation. If you would like to install RobotStudio without installing PC SDK, or remove PC SDK from RobotStudio select *Custom* in the RobotStudio installation wizard and select or unselect the feature PC SDK



NOTE!

You are also strongly advised to study the Release Notes that you will find on the RobotWare DVD and on the RobotStudio Community web site, as these hold the most up-to-date information, including new features and any known limitations of the release.



NOTE!

Please note that there is no license key.

© Copyright 2010 ABB. All rights reserved.

2.2. How to obtain and install a license key for RAB 5.09 or earlier

Overview

In RAB 5.10 the license check was removed from the software, which allows anyone to use Robot Application Builder for free. This means you no longer need to bother about getting a license, or including a licx file in your PC application.

NOTE!

For RAB version 5.09 or earlier, licensing is the second part of the installation procedure. In case you need to develop a RAB application for RobotWare 5.09 or earlier you need to turn to support to get a free license key.

Install licence key

Follow these steps when you have received the e-mail with the license key file:

Step	Action
1	Detach the license key file from the e-mail and save it to a folder on your PC.
2	Double-click the license key file. This opens the **License Install Wizard**.
3	Follow the instructions in the wizard.

NOTE!

To execute controller applications towards a real robot controller you must connect your PC to the robot controller, either through the network or directly to the service port on the controller. For detailed information, see *How to set up your PC to communicate with robot on page 23*.

© Copyright 2010 ABB. All rights reserved.

2.3. How to set up your PC to communicate with robot

Overview

This section describes how to connect your PC to the robot controller.

You can either connect the PC to the controller through an Ethernet network or directly to the controller service port. When using the controller service port, you can either obtain an IP address for the PC automatically, or you can specify a fixed IP address.

When the PC and the controller are connected correctly, the controller is automatically detected by RobotStudio.

NOTE!

A PC SDK application requires RobotStudio or ABB Robot Communications Runtime to connect to a controller in run-time. The latter is included in the RobotStudio installation.

Why is a connection needed?

Connecting the PC to the controller is necessary for all online tasks performed in RobotStudio. For example, downloading a robot system or files to the controller, editing configuration files, programming and so on.

It is necessary for executing PC applications targeting a real robot controller.

It also enables you to communicate with the controller by means of a console window on the PC and get valuable information about controller status, FlexPendant memory consumption and so on.

Ethernet network connection

If the controller is connected to an Ethernet network, you can connect the PC to that network as well. The settings to use on the PC depends on the network configuration. To find out how to set up your PC, contact the network administrator.

Service port connection with automatic IP address

An alternative to network connection is using the controller service port. It has a DHCP server that automatically gives your PC an IP address if it is configured for this. For more information about configuring the PC to obtain an IP address automatically, see *Windows Help* on *Configure TCP/IP settings*.

NOTE!

Obtaining an IP address automatically might fail if the PC already has an IP address from another controller or Ethernet device. To make sure that you get a correct IP address if the PC has already been connected to an Ethernet device, do one of the following:

- Restart the PC before connecting to the controller.
- Run the command “ipconfig /renew” from the command prompt after connecting the PC to the controller

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

Service port connection with fixed IP address

Instead of obtaining an IP address automatically, you can specify a fixed IP address on the PC you connect to the controller.

Use the following settings for connecting with a fixed IP address:

Property	Value
IP address	192.168.125.2
Subnet mask	255.255.255.0
Default Gateway	192.168.125.1

Related information

For information about	See
How to set up PC network connections	*Windows Help - Configure TCP/IP settings*.
How to connect the PC to the Controller service port	*Connecting a PC to the Service Port* in the RobotStudio help.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

2.4. Development environment

Overview

This section presents an overview of the development environment used to create PC SDK applications. The application has to be programmed and debugged using Microsoft Visual Studio.

Microsoft .NET and Microsoft Visual Studio

Microsoft Visual Studio is supported by the .NET Framework. A core component of the .NET Framework is the common language runtime (CLR). It manages code execution, threads and memory, while also enforcing type safety.

Another major component is the Base Class Library, which is a comprehensive, object-oriented collection of reusable types. To become a skilled .NET programmer it is essential to learn the functionality offered by the Base Class Library.

It is not in the scope of this manual to teach how to use Visual Studio. For this purpose msdn (Microsoft Developer Network) at http://msdn.microsoft.com is a useful source of information.



NOTE!

From PC SDK 5.11 Visual Studio 2008 is also supported. For information about upgrading an existing PC SDK project to Visual Studio 2008, see *Conversion of VS 2005 projects to Visual Studio 2008 on page 29*.

Choosing a programming language

Together with Visual Basic, C# is the most widely used .NET language.

C# is an object-oriented language derived from C, with some features from C++, Java and Visual Basic. It was designed for .NET and offers the power and richness of C++ along with the productivity of Visual Basic. Both PC and FlexPendant SDK are implemented using C#.

For PC SDK applications any of the .NET languages can be used. ABB support, however, is offered only in C# and Visual Basic.NET. Likewise, in this manual there are code samples in C# and Visual Basic.NET, but none in J# or Visual C++.

At run-time it does not matter which language you have used, as compiled .NET code is language independent. The source code compiles from a high-level language into Intermediate Language (IL), which is then executed, at runtime, by the Common Language Runtime. This makes it possible to use different programming languages, even within the same application. For more information on .NET terms, see *Definitions on page 17*.



NOTE!

It is presumed that you are already a .NET programmer. If not, you need to start by learning the programming language to be used. There are numerous books teaching C# and Visual Basic.

© Copyright 2010 ABB. All rights reserved.

Integration with Visual Studio

PC application uses the standard design support. As you will see, using PC SDK is quite intuitive for a developer used to Visual Studio programming.



NOTE!

The help module is not integrated with the Visual Studio Help function. Clicking F1 when pointing at code, for example, will open the *Visual Studio Programmer's Reference* or the *.NET Framework Class Library* for the specific language and topic. If your problem is PC SDK related this will not help you.



TIP!

Depending on what kind of application you are working at, locate the *Reference Manual PC SDK*. Click **Start** menu, point to **Programs, ABB Industrial IT, Robotics IT, RobotStudio 5.xx/SDK**. Keep the reference file open while programming, as you will be needing it all the time.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

2.5. Two development models - virtual and real

About this section

When trying out a custom application, you can either use a virtual robot controller or a real robot system. This section provides information on how to use both the development models.

Virtual robot technology

The virtual IRC5 of ABB's RobotStudio allows the IRC5 controller software to execute on a PC, and supports application developers with a purely virtual environment to be used for development, test and debug.

When you start the virtual IRC5 in RobotStudio, a virtual robot cabinet along with a virtual FlexPendant appears on the PC screen.

As a real robot controller is normally not readily at hand for application development, virtual technology is very valuable.

Requirements for virtual environment

The following software components must be installed to develop, test and debug using the virtual environment:

- ABB RobotStudio (Complete or Custom - with RobotStudio and PC SDK selected)
- Microsoft Visual Studio 2005 Express or better, or, Microsoft Visual Studio 2008 Express or better.

The RobotWare option *PC Interface* is not needed in the virtual environment.



NOTE!

For more information, see *Installing and Licensing RobotStudio* in *Operating Manual - RobotStudio*

Requirements for real environment

The following software components must be installed to develop, test and debug using a real robot controller:

- ABB RobotStudio (Complete or Custom - with RobotStudio and PC SDK selected)
- Microsoft Visual Studio 2005 Express or better, or, Microsoft Visual Studio 2008 Express or better.
- Controller option PC Interface
- Network connection between PC and robot controller

For information about how to set up the network, see *How to set up your PC to communicate with robot on page 23*.

Virtual test and debug

A PC application will run as an independent executable (exe). Using the virtual environment, it targets the virtual IRC5 instead of a real robot controller.

Debugging is easy using the virtual IRC5 and Visual Studio. You attach the application process to Visual Studio, set a break point in the code, and step through it as it executes.

Continues on next page

© Copyright 2010 ABB. All rights reserved.

Continued

Real tests necessary

The virtual environment is a very convenient choice, especially for testing and debugging.

This means that potential problems may be hard to detect until you test the application using a real robot system.

Porting the application from virtual to real IRC5

As for a PC SDK application, you will hardly notice any difference when using it with a real IRC5 controller. The only real difference is that the communication between the application and the controller will now be done over a network, which may have an impact on performance.

Deployment to customer

During development, deployment to the controller is done manually. When the development phase is over and the application needs to be deployed to the customer, this should be done differently.

For information about how this should be done, see *Deployment of a PC SDK application on page 133*.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

2.6. Conversion of VS 2005 projects to Visual Studio 2008

Overview

Converting an existing PC SDK Visual Studio 2005 project to Visual Studio 2008 is simple. When you open a VS 2005 project in VS 2008, the Visual Studio Conversion Wizard appears automatically. The procedure which converts the project to VS 2008 is easy to follow. It consists of a few dialog box providing information about what will happen.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

2.6. Conversion of VS 2005 projects to Visual Studio 2008

Continued

© Copyright 2010 ABB. All rights reserved.

Continues on next page

3 Run-time environment

3.1. Overview

About this chapter

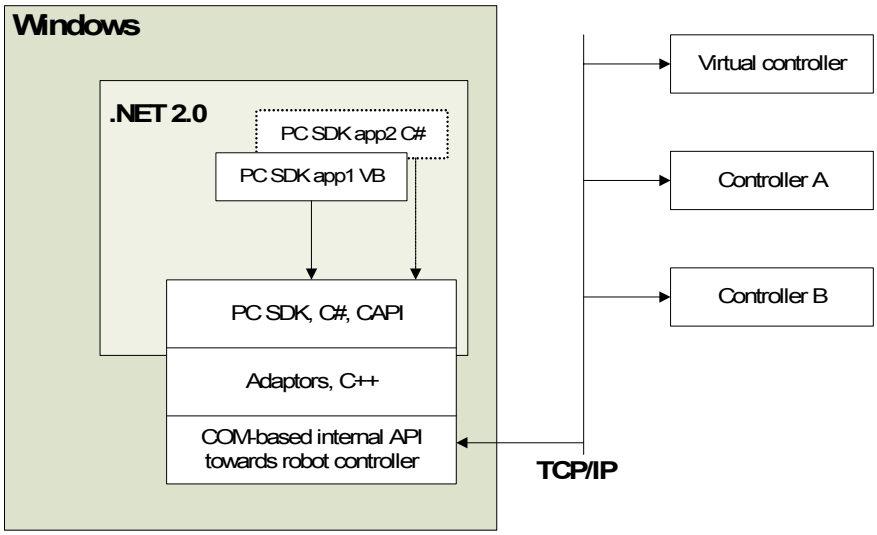
This chapter provides an overview of the run-time environment of custom applications, including illustrations of the software architecture of the PC SDK.

This explains how communication is carried out between the client and the robot controllers, as well as how clients access controller resources.

Software architecture

The following illustration shows the Software archtitecture of PC SDK. It shows the PC platform. Two PC SDK applications developed on top of the PC SDK. The PC SDK CAPI is the public API offering controller functionality. A PC SDK application can control many robot controllers on the network. All communication with these is done via the internal Robot Communication Runtime.

PC platform



3.1_1

Controller API

The PC SDK offer controller functionality through the public application interface called Controller API. The interface can be seen as a specification of the controller services available.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

3.2 Running PC SDK Applications

3.2.1. Overview

Introduction

A PC SDK application runs as a .NET executable, started either by double clicking the exe-file or by browsing to the program using the Windows **Start** menu.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

3.2.2. Mastership

Controlling controller resources

Controller resources must be controlled by a single client at a time. Several people can be logged on to the controller, but only one person at a time can run commands or change RAPID data. This is for security reasons as well as for protecting data from being accidentally overwritten.

When logged on to a controller you can have either read-only access or write access. Read only is the default access right. To get write access the client needs to request mastership of the controller resource it wants to manipulate.

NOTE!

In addition to the access right system, there is the User Authorization System, which restricts what each user is allowed to do with the robot. For more information, see *User Authorization System on page 46*.

Manual and automatic mode

When the controller is in manual mode, the FlexPendant has priority to write access. Mastership will not be given to a remote client unless an operator explicitly allows this through the FlexPendant. At any time, the operator can click *Revoke* on the FlexPendant to get the write access back.

In automatic mode, the client who first requests write access will get it. If a remote client has taken mastership of a domain other remote clients will not be allowed write access, but will get an exception if they try. For the operator, there is no way to revoke mastership to the FlexPendant, but to switch the operating mode of the controller to manual.

As for a remote client, such as a PC SDK application, however, mastership handling has to be carefully implemented by the application programmer.

Controller mastership domains

The following Controller domains require mastership:

- Rapid
- Configuration

For code examples, see *Start program execution on page 59* in the PC SDK section.

NOTE!

Operations that require mastership are more resource demanding. Mastership should therefore be released as soon as an operation is completed.

© Copyright 2010 ABB. All rights reserved.

Remote privilege in manual mode

Most of the time, it is unconvenient to have a PC SDK application perform operations that require mastership when the controller is in manual mode. Starting program execution, for example, is not even possible.

In manual mode when a remote client, for example RobotStudio or a PC SDK application, requests mastership a dialog box will appear on the FlexPendant. It enables the operator to grant mastership to the requesting client.

If mastership is granted, the remote application has the privilege to access robot controller resources. Meanwhile, the FlexPendant is locked and cannot be used until the remote application releases mastership or mastership is lost for any of the reasons mentioned in *Losing mastership on page 34*.

Losing mastership

Remote clients loses the mastership without warning for the following reasons:

- change from automatic to manual mode
- controller restart
- lost communication
- in manual mode forced revocation of mastership by another client with higher priority - for example the FlexPendant

If mastership is lost, it has to be taken back explicitly by the client. The controller does not store the information.



NOTE!

The FlexPendant may also lose mastership without any warning. This may happen in automatic mode, when a RobotStudio user or a PC SDK application asks for write access to the controller, for example. The status bar of the FlexPendant will then indicate *“Remote Access in Auto”*.

© Copyright 2010 ABB. All rights reserved.

3.2.3. PC application configuration

Application configuration file

All .NET Winform applications are designed to read configuration from an *App.config* file in the application directory. It is not mandatory to use such a file in a PC SDK application, but it is sometimes a handy way to add application flexibility.

For your convenience an *App.config* file that you can use is included in the PC SDK installation. The default values, which the PC SDK uses if there is no configuration file to read, are entered in the file. To modify application behavior you thus need to change the values of the attributes of this file.



NOTE!

Even if you use the App.config file to specify which controllers to work with you must still use the netscan functionality to establish a connection from your PC application. For more information, see *Discovery domain on page 61*.

Adding App.config to the project

Start by copying the *App.config* file at C:\Program Files\ABB Industrial IT\Robotics IT\SDK\PC SDK 5.xx to the directory of your .csproj file. Then add the file to the project by right-clicking the project icon, pointing to **Add** and selecting **Existing Item**.



NOTE!

The *Copy Local* property of the PC SDK references used by your application must be set to true to make use of the App.config file. (In the *Solution Explorer* in Visual Studio, right-click the reference and select *Properties*.)

Section tag

The `<section>` tag in the `<configSection>` part of the *App.config* should specify that there is a *capi* section in the file. It should also specify which type is to be used as the configuration container object as well as the assembly that this type belongs to:

```
<section name="capi"
     type="ABB.Robotics.Controllers.ConfigurationHandler,
     ABB.Robotics.Controllers"/>
```

Capi section

The PC SDK application specific configuration data should be added to the `<capi>` section.

```
<capi>
  ...
</capi>
```

© Copyright 2010 ABB. All rights reserved.

The following tags are implemented in the PC SDK:

<defaultSystem>

If there is a controller (robot system) in the network that you connect to often, you may want to use the `<defaultSystem>` tag. It has an `id` attribute containing a string embraced by curly brackets. It is the system's globally unique identifier (GUID), which can be find in the *system.guid* file in the INTERNAL folder of the robot system.

```
<defaultSystem id="{469F56DF-938E-4B06-B036-AABBB3E61F83}" />
```

Using this mechanism enables you to use the default constructor to create a `Controller` object for the specified controller:

```
VB:
Dim aController As Controller = New Controller()

C#:
Controller aController = new Controller();
```

<remoteControllers>

It is possible to add controllers outside the local network when scanning for available controllers. One way of doing that is to add the IP address of these controllers in the `<remoteControllers>` tag:

```
<remoteControllers><controller id="192.168.0.9" />
<controller id="192.168.0.19" />
</remoteControllers>
```

<discovery.networkscanner>

You can configure how long (in ms) a scan operation will last, and increase the value if netscanning fails. The default value is 200, but if you have a slow PC longer time might be needed.

```
<discovery.networkscanner delay="400" />
```

<defaultUser>

The `<defaultUser>` tag holds information about user name, password and an optional application name for the default user. It is used by the `UserInfo` class to log on to a controller. If an application name is not supplied, the process name is used.

```
<defaultUser name="user name" password="password"
     application="application"/>
```

<rmmp>

When mastership is requested in manual mode by a remote client such as RobotStudio or a PC SDK application, a dialog is launched on the FlexPendant asking the operator to confirm that mastership should be passed from the FlexPendant to a remote client. As long as there is no confirmation on the FlexPendant the PC SDK application is not given mastership. The time-out parameter is the time in seconds that the PC SDK application will wait for someone to confirm remote access in the FlexPendant dialog. The cycle parameter is the time in ms between poll calls from the PC SDK to check whether mastership has been granted.

```
<rmmp cycle="550" timeout="65" />
```

© Copyright 2010 ABB. All rights reserved.

<controllerCall>

You can add a time-out in ms and a multiplicand for slow calls to the controller. The time-out parameter is the maximum time a call through the Controller API will be permitted. If no answer is returned within the time specified, an exception is thrown. A slow call is a call that takes longer than usual, usually operations which require a UAS grant:

```
<controllerCall timeout="27000" slow="2.1" />
```

<eventStrategy>

The default way to handle events from the controller is to use asynchronous delegates (`AsyncDelegate`), applying the `Invoke` method to synchronize events and GUI.

By using an `<eventStrategy>` tag, you can choose to use a windows postback delegate instead. To make this work you must also implement a subscription to the event from a windows form, or else the event handler will not receive the event:

```
<eventStrategy name="WindowDelegate" />
```



NOTE!

Using this strategy for event handling may affect the performance of your application.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

3.2.4. Communication between PC and controller

COM technology

The PC SDK uses an internal Controller API based on COM technology to communicate with the controller. This API uses sockets and the local TCP/IP stack (see *Definitions on page 17*) towards both real and virtual controllers.



NOTE!

You should be aware that the .NET garbage collector does not collect COM objects, but these need to be disposed of explicitly by the application programmer. For more information, see *Memory management in PC applications on page 63*.

Resource identification

All controller resources, both real and virtual, are described using object based hierarchy. Each resource is uniquely identified, including information about which controller owns the resource by use of the unique system id or IP address of the controller.

The controller is the top object of the hierarchy:

```
"/<Controller>/<Domain>/<SubDomain1>/<SubDomain2>/and so on"
```



TIP!

Error messages including such a path indicate where to look for the problem.

Hard real-time demands

The PC CAPI cannot meet hard real-time demands for the following reasons:

- Part of the API executes on a non-real-time operating system.
- Communication is performed with TCP/IP over a network
- The controller sometimes has tasks to perform that have a higher right of priority.



NOTE!

A minimum response time for real controllers should be expected to be in the order of 10-100 milliseconds.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Continued

3.2.5. Licence verification - applies only to versions earlier than PC SDK 5.10

Overview

Deployed PC applications does the license verification during execution, checking that all application assemblies have been built on a PC with a valid PC SDK license key. If the key is missing some functions in the PC SDK will raise an exception during execution.



NOTE!

The license verification was removed in the 5.10 release. So the licx file detailed in the next section is no longer needed.

Licenses.licx

The license key should be placed in a “Licenses.licx” file, which should be added to your project as an embedded resource. For your convenience, such a file is included in the PC SDK installation at C:\Program Files\ABB Industrial IT\Robotics IT\SDK\PC SDK 5.xx. The key for the PC SDK in VS 2005 is:

```
ABB.Robotics.Controllers.Licenses.PCSdk, ABB.Robotics.Controllers
```



NOTE!

The preceding path applies to RAB 5.09 and earlier versions. It does not apply for the new installations.



TIP!

The License Compiler (Lc.exe) is a .Net Framework tool. It generates a .license file from a .licx file. Search in MSDN (licx , lc.exe) if you want more detailed information.

© Copyright 2010 ABB. All rights reserved.

3.3. Release upgrades and compatibility

About this section

This section addresses compatibility issues with different version of Robotware.



NOTE!

PC SDK 5.10 supports Visual Studio 2005. PC SDK 5.11 to 5.13 supports VS 2005 and VS 2008.

Matching PC SDK and RobotWare release

You should be aware that the PC SDK are developed and tested for a specific RobotWare release. The general rule is therefore that you develop an application for a certain release.

Compatibility between revisions is however guaranteed (for example RobotWare 5.11.01 will be compatible with PC SDK 5.11).

RobotWare upgrades

At some time during the lifetime of your application, a robot system that your application targets may be upgraded with a later RobotWare version.

The PC SDK it is normally compatible with a newer RobotWare release. The PC that hosts the PC SDK application at the customer, however, still needs an upgrade of the Robot Communication Runtime, so that it matches the new robotware release. For more information, see *ABB Industrial Robot Communication Runtime.msi on page 133*. If you decide to upgrade the PC SDK application, you must also remember to upgrade the runtime environment of the customer's PC. For more information, see *Deployment of a PC SDK application on page 133*.



NOTE!

You find all the details about compatibility between different PC SDK versions in the Release Notes.



TIP!

When compiling your project, notice any warnings of obsolete methods, as these will probably be removed in the next PC SDK release.

Prepared for change

To summarise it is important to keep source code safe and available for maintenance.



TIP!

Ildasm is a Microsoft tool, which comes with the installation of Visual Studio, that you may find useful. It enables you to open the manifest of a specified assembly and quickly find out about dependencies for example.

Find out more about it at http://msdn2.microsoft.com/en-us/library/aa309387(VS.71).aspx

Continues on next page

© Copyright 2010 ABB. All rights reserved.

4 Developing Controller applications

4.1. Introduction

About this chapter

This chapter deals with analysis, design, and implementation of PC SDK applications.

It provides the following specific programming issues that are important for PC SDK users:

- thread conflicts and how to avoid them
- controller events and event handling
- errors and exception handling
- the User Authorization System

The chapter does not include hands-on information on how to set up your first project or detailed information on how to use the PC SDK class libraries, as these topics are covered in dedicated chapters.

Basic approach

In most aspects, using the PC SDK for application development presents no major difference compared to ordinary .NET development. The .NET class libraries are used for everything that is not robot specific. In addition, you use the public Controller API of the SDKs.

Step	Action
1.	**Before you start** Learn the basics about PC SDK programming by reading all relevant sections of this manual. Feel reassured that this is a timesaving activity and do not rush into coding.
2.	**During development** Frequently test application functionality.

© Copyright 2010 ABB. All rights reserved.

4.2. Analysis and design

About this section

The purpose of PC SDK is to provide operator interfaces that fulfill specific customer needs. This section focusses on the development phases preceding the actual coding: analysis and design.

Object oriented software development

.NET is entirely object-oriented. Platform services are divided into different namespaces such as `System.Collections`, `System.Data`, `System.IO`, `System.Security` and so on. Each namespace contains a set of related classes that allow access to platform services. PC SDK, too, is completely object oriented. Its class libraries are organized in different namespaces such as `ABB.Robotics.Controllers.RapidDomain`, `ABB.Robotics.Controllers.MotionDomain` and so on

You need to have some experience in object orientation is necessary to start developing custom applications. It is presumed that you feel comfortable with concepts such as objects, classes, methods, inheritance, encapsulation and so on.

Object oriented Analysis and Design

Object Oriented Analysis and Design, OOAD, is a popular topic in computer science literature, where the importance of doing a thorough analysis and design before starting coding is commonly accepted. A well designed OO application has a true representation in the real world. Classes have well defined areas of responsibility and collaborate in an efficient way to achieve what is required.

Analysis based on communication and use cases

The main idea of PC SDK is, as has already been pointed out, that custom operator interfaces can be developed close to end-users, taking their specific needs in consideration. It therefore goes without saying that analysis is crucial.

The result of the object-oriented analysis is a description of what we want to build, often expressed as a conceptual model. Any other documentation that is needed to describe what we want to build, for example pictures of the User Interface, is also part of analysis.

The most important aspect for PC SDK development is communication with end-users. Activities which support a shared view of what should be developed are strongly recommended. Such activities may include:

- creating and discussing use cases together
- coding or drawing prototypes and get end-user feedback

In short, involving end-users from the early stages and keeping them involved throughout the development project is the best strategy.



NOTE!

Customer satisfaction is what has driven the development of PC SDK. Do make sure that you have really understood what the end-users of your application need.

© Copyright 2010 ABB. All rights reserved.

Design is about managing complexity

The result of the object-oriented design details how the system can be built, using objects. Abstraction is used to break complex problems into manageable chunks. It makes it possible to comprehend the problem as a whole or to study parts of it at lower levels of abstraction.

It takes years to become a skilled object oriented designer. Design theory must be transformed into practical experience and iterated over and over again.

The goal is to produce high quality code, which is cost-efficient and easy to maintain. This is achieved, for example, when adding new functionality will involve minimal changes to existing code and most changes will be handled as new methods and new classes.

Do you need to do design?

There is a huge difference in complexity when creating software such as .NET framework, for example, and a custom operator view for IRC5. Obviously, the more complex a system the more careful design is needed. Accordingly, the larger and more complex a custom application needs to be, the more likely you are to spend time on design.

This table presents some considerations before deciding how well you need to design your application before starting coding:

Consideration	Advice
How much code is it going to be?	If it is going to be a very simple application with just one view and a few buttons there is no need even to split the code between different classes and files. If there will be a substantial amount of code and there might be further extensions later on, spending time on design becomes more relevant.
Will different developers work on different classes/ components? Will you maintain the code yourself, or may it be done by others?	If yes, spending time on design becomes more relevant.
Is the real time aspect of the application important?	If yes, coding efficiently is important. This will much more easily be achieved if you spend some time on design.

As complex or as easy as you wish

A simple custom application can be created in a day or two using PC SDK. A large custom application with a number of different views, offering advanced robot system functionality, however, may take months to develop and will require considerable programming skill.The recommendation is to start developing a simple application, which you execute on the target platform, before moving on to advanced PC SDK programming.

© Copyright 2010 ABB. All rights reserved.

4.3. Controller events and threads

Overview

A controller event is a message from the controller that something has happened. Events can be caught and acted upon by PC SDK applications.

Controller events use their own threads. This means that user interface threads and controller event threads can get into conflict. This section gives information on how to prevent this.

Controller events

PC SDK applications can subscribe to a number of controller events. These are all described in the *Reference Manual PC SDK*.

The following table shows some events that exists in the PC SDK.

The event...	occurs when...
StateChanged	the controller state has changed.
OperatingModeChanged	the controller operating mode has changed.
ExecutionStatusChanged	the controller execution status has changed.
Changed	the value or the state of the I/O signal has changed.
MessageWritten	the EventLog has a new message
ValueChanged	the value of a RAPID data has changed.

NOTE!

There is no guarantee you will get an initial event when setting up/activating a controller event. You need to read the initial state from the controller.

GUI and controller event threads in conflict

You should be aware that controller events use their own threads on the PC platform. If a GUI thread and a controller event thread get into conflict, deadlocks or overwritten data may occur. This may happen when a controller event is succeeded by an update of the user interface, which is indeed a very common scenario.

You then need to take action in your code to control that the user interface update is executed by the GUI thread and not by the controller event thread. This is done by enforcing a thread switch using the `Invoke` or `BeginInvoke` method. For more information with code examples, see *Invoke method on page 45*.

On the other hand, if the controller event should NOT cause any update of the user interface, you should not take any special action. Using `Invoke` / `BeginInvoke` is performance demanding and should not be used more than necessary.

NOTE!

Thread conflicts often cause hangings. The PC application UI then stops responding and the application has to be restarted.

Examine what exception has been thrown when you encounter such a situation. The exception `System.InvalidOperationException` (PC platform) indicate thread conflicts. See the next section for information on how to use `Invoke` to solve the problem.

© Copyright 2010 ABB. All rights reserved.

Invoke method

All PC application windows views must inherit `Control` / `TpsControl`, which implement `Invoke` and `BeginInvoke`. These methods execute the specified delegate/event handler on the thread that owns the control's underlying window handle, thus enforcing a switch from a worker thread to the GUI thread. This is precisely what is needed when a controller event needs to be communicated to the end user of the system.

`Invoke` should be called inside the event handler taking care of the controller event. Notice that you have to create a new object array for the sender and argument objects:

VB:

```
Me.Invoke(New EventHandler(UpdateUI), New [Object]() {Me, e})
```

C#:

```
this.Invoke(new EventHandler(UpdateUI), new Object[] { this, e });
```

Also notice that if you use `EventHandler` in the `Invoke` method and not the specific delegate class, for example`DataValueChangedEventHandler`, you need to typecast the argument in the delegate which updates the user interface. How this is done is shown by the example below:

VB:

```
Private Sub UpdateUI(ByVal sender As Object, ByVal e As EventArgs)
   Dim args As StateChangedEventArgs
      args = DirectCast(e, StateChangedEventArgs)
      Me.label1.Text = args.NewState.ToString()
End Sub
```

C#:

```
private void UpdateUI(object sender, EventArgs e)
        {
  StateChangedEventArgs args;
  args = (StateChangedEventArgs)e;
  this.label1.Text = args.NewState.ToString();
}
```



NOTE!

The difference between `Invoke` and `BeginInvoke` is that the former makes a synchronous call and will hang until the GUI operation is completed, whereas `BeginInvoke` executes the specified event handler asynchronously. Which method you want to use depends on the logics of your program. The recommendation is to choose `BeginInvoke` whenever possible.



NOTE!

If your code tries to access a GUI control from a background thread the .NET common language runtime will throw a `System.NotSupportedException` (FlexPendant platform) or a `System.InvalidOperationException` (PC platform).

© Copyright 2010 ABB. All rights reserved.

4.4. User Authorization System

Overview

In the robot controller there is a system controlling user access: the *User Authorization System (UAS)*. If this feature is used each user needs a user name and a password to log on to a robot controller via RobotStudio. If the controller connection for any reason is lost, you need to log on again.

The controller holds information on which operations different users are allowed to perform. The UAS configuration is done in RobotStudio.



TIP!

To learn more about UAS use the help function in RobotStudio.

Accessing UAS from custom applications

Accessing UAS is done by using the property `AuthorizationSystem` on the controller object:

VB:

```
Dim uas As UserAuthorizationSystem =
     aController.AuthenticationSystem
```

C#:

```
UserAuthorizationSystem uas = aController.AuthenticationSystem;
```

Grants and Groups

UAS rights are called *Grants*. The specific user belongs to one of several defined *Groups*, where each group has a number of specified grants.

To ensure that you have the necessary grant to perform an operation, you use the `CheckDemandGrant` method on the `AuthorizationSystem` object. The grant to check is passed as an argument:

VB:

```
If uas.CheckDemandGrant(Grant.LoadRapidProgram) Then
  aTask.LoadModuleFromFile(localFile, RapidLoadMode.Replace)
End If
```

C#:

```
if (uas.CheckDemandGrant(Grant.LoadRapidProgram))
   {
  aTask.LoadModuleFromFile(localFile, RapidLoadMode.Replace);
  }
```



NOTE!

The PC SDK application cannot override the UAS configuration. This means that the system will in the end prevent you from performing an action that is not allowed.

© Copyright 2010 ABB. All rights reserved.

MessageBox feedback

If a UAS grant is missing, you should be informed about it. This can be done in a message as shown in this example:

```
msg = "You are not allowed to perform this operation, talk to your
     system administrator if you need access."
title = "User Authorization System"

MessageBox.Show(msg, title, MessageBoxButtons.OK,
     MessageBoxIcon.Exclamation);
```

GetCurrentGrants and DemandGrant

Another possibility is to retrieve all grants for the current user calling `GetCurrentGrants`, then iterate over the grants collection and search the necessary grants.

Yet another solution is to call `DemandGrant` with one of the static `Grant` members as in argument.

If you do not have the specified grant, the PC SDK throws a `GrantDemandRejectedException`.



TIP!

Learn more about UAS and `Grant` members in the *Reference Manual PC SDK*.

© Copyright 2010 ABB. All rights reserved.

4.5. Exception handling

Overview

The .NET programming languages provide built-in support for exception handling, which allows the program to detect and recover from errors during execution.

In managed code, execution cannot continue in the same thread after an unhandled exception. The thread terminates, and if it is the program thread, the program itself terminates. To avoid this, accurate exception handling should be used.

Exceptions thrown from the controller are handled by the PC SDK `ExceptionManager`, which converts the internal HRESULT to a .NET exception with a reasonable exception description, before it is thrown to the custom application layer. The application handling of these exceptions should apply to general .NET rules.

Exceptions are expensive in a performance perspective and should be avoided if there are other alternatives. If possible use a `try-finally` block to clean up system and unmanaged resource allocations.

Try-catch-finally

Exceptions are handled in `try` - `catch` (-`finally`) blocks, which execute outside the normal flow of control.

The `try` block wraps one or several statements to be executed. If an exception occurs within this block, execution jumps to the `catch` block, which handles the exception.

The `finally` block is executed when the `Try` block is exited, no matter if an exception has occurred and been handled. It is used to clean up system or controller resources.

If you do not know what exceptions to expect or how to handle them, you can catch them and do nothing. This, however, may result in difficult error tracing, as exceptions include information on what caused the problem. Therefore, try at least to display the exception message, either by using a message box or the types `Debug` or `Trace`.

Typecasting

When typecasting `Signal` or `RapidData` values, for example, there is a potential risk of typecast exceptions. To avoid this you can check the object using the `is` operator for both value and reference types:

VB:

```
If TypeOf aRapidData.Value Is Num Then
  Dim aNum As Num = DirectCast(aRapidData.Value, Num)
  ......
End If
```

C#:

```
 if (aRapidData.Value is Num)
  {
                  Num aNum = (Num)aRapidData.Value;
  ......
              }
```

© Copyright 2010 ABB. All rights reserved.

In C# it is also possible to use the `as` operator for reference types. A null value is returned if the type is not the expected one:

C#:

```
DigitalSignal di = this.aController.IOSystem.GetSignal("UserSig")
     as DigitalSignal;
if (di == null)
{
  MessageBox.Show(this, null, "Wrong type");
}
```

.NET Best Practices

.The *.NET Framework Developer's Guide* presents the following best practices for exception handling:

- Know when to set up a try/catch block. For example, it may be a better idea to programmatically check for a condition that is likely to occur without using exception handling. For errors which occur routinely this is recommended, as exceptions take longer to handle.
- Use exception handling to catch unexpected errors. If the event is truly exceptional and is an error (such as an unexpected end-of-file), exception handling is the better choice as less code is executed in the normal case.Always order exceptions in catch blocks from the most specific to the least specific. This technique handles the specific exception before it is passed to a more general catch block.

© Copyright 2010 ABB. All rights reserved.

4.6. How to use the online help

Overview

The online help is available along with the installation of PC SDK and is accessible from Windows **Start** menu.

You are recommended to read this Application manual carefully as you develop your first PC SDK application. *PC SDK Reference* is an important complement to this manual, as these make up the complete reference to the class libraries of PC SDK. For more information, see *Documentation and help on page 15*.



NOTE!

The *SDK Reference* is NOT integrated in Visual Studio. You must access it from the **Start** menu.



TIP!

For more information on the web address to *RobotStudio Community*, where PC SDK developers discuss software problems and solutions online, see *Documentation and help on page 15*.

© Copyright 2010 ABB. All rights reserved.

5 Using the PC SDK

5.1. Controller API

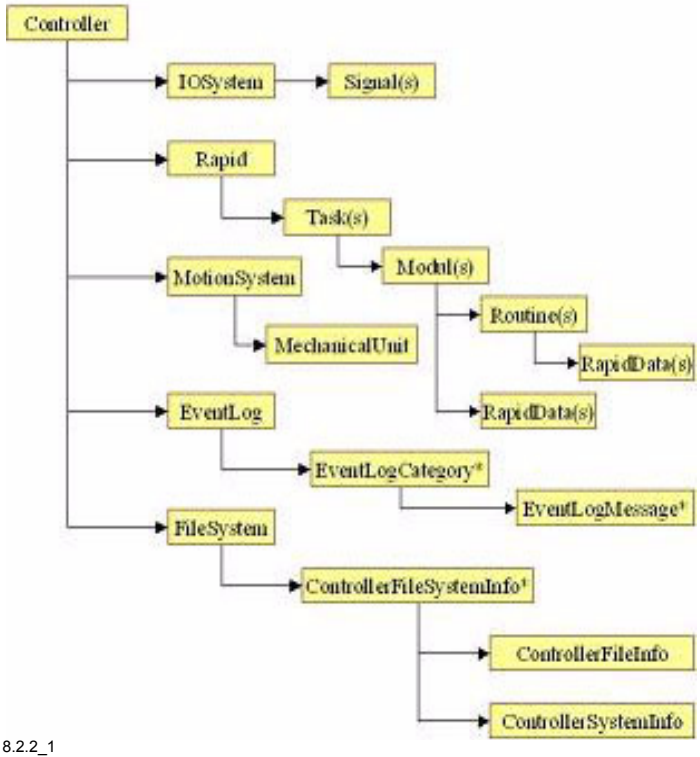
PC SDK domains

The PC SDK class libraries are organized in the following domains:

- Controllers
- ConfigurationDomain
- Discovery
- EventLogDomain
- FileSystemDomain
- Hosting
- IOSystemDomain
- Messaging
- MotionDomain
- RapidDomain
- UserAuthorizationManagement

CAPI illustration

The classes used to access robot controller functionality together make up the *Controller API* (CAPI). The following illustration shows a part of the CAPI object model:



8.2.2_1

© Copyright 2010 ABB. All rights reserved.

PC SDK Reference

This Application manual covers some of the PC SDK functionality, but is by no means a complete guide to the APIs of the PC SDK.

The *Reference Manual PC SDK* is the complete reference of the PC SDK class libraries. It should be your companion while programming.

It can be launched from Windows **Start** menu by pointing at *Programs* - *ABB Industrial IT* - *Robotics IT* - *RobotStudio 5.xx* - *SDK* and selecting *PC SDK Reference*.

© Copyright 2010 ABB. All rights reserved.

5.2. Create a simple PC SDK application

Overview

To get started with programming, create a simple application that displays all the virtual and real controllers on the network. It should then be possible to log on to a controller and start RAPID execution.



CAUTION!

Remote access to controllers must be handled carefully. Make sure you do not unintentionally disturb a system in production.

Setting up the project

Use this procedure to set up a PC SDK project:

Step	Action
1.	On the **File** menu in Visual Studio, select **New** and click **Project**. Select a **Windows Application** project.
2.	Add the references to the PC SDK assemblies, *ABB.Robotics.dll* and *ABB.Robotics.Controllers.dll*, to the project. The assemblies are located in the installation directory, by default at C:\Program Files\ABB Industrial IT\Robotics IT\SDK\PC SDK 5.xx.

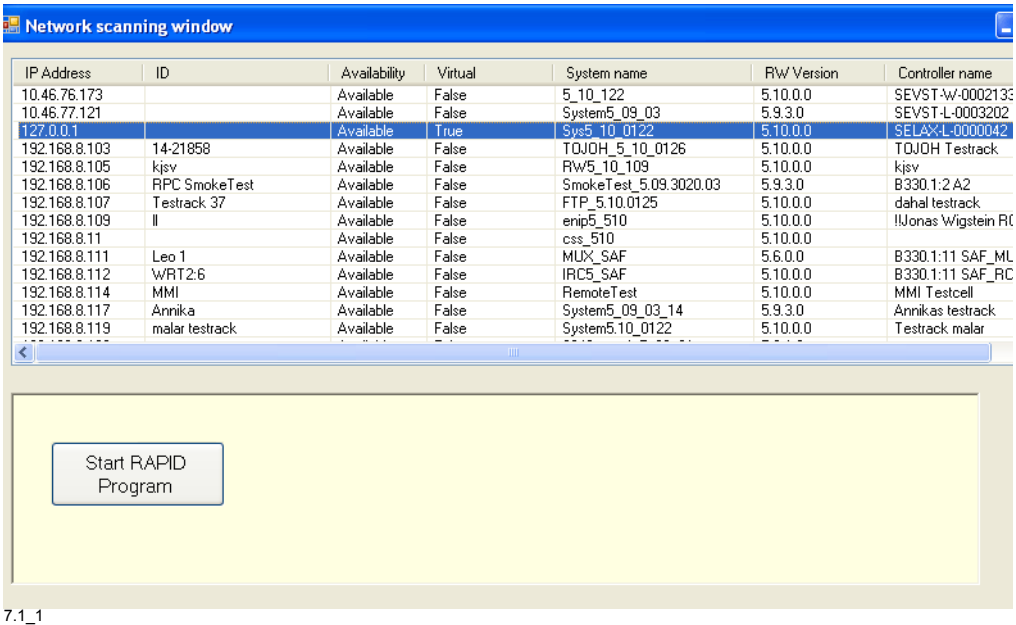
3.	Open *Form1.cs* and add the needed namespace statements at the top of the source code page:
	VB:
	`Imports ABB.Robotics`
	`Imports ABB.Robotics.Controllers`
	`Imports ABB.Robotics.Controllers.Discovery`
	`Imports ABB.Robotics.Controllers.RapidDomain`
	
	C#:
	`using ABB.Robotics;`
	`using ABB.Robotics.Controllers;`
	`using ABB.Robotics.Controllers.Discovery;`
	`using ABB.Robotics.Controllers.RapidDomain;`

4.	In the **Solution Explorer** right-click *Form1.cs* and select **View Designer**. Create the Graphical User Interface according to the instruction in the next section.

© Copyright 2010 ABB. All rights reserved.

Create the user interface

The following screenshot shows the running PC SDK application that we will create. As you see both virtual and real controllers on the network are included in a network scan.



7.1_1

Use this procedure to create the user interface of the application:

Step	Action
1.	Change the **Text** property of the form to “Network scanning window”.
2.	Change its **Size** to 850; 480.
3.	Add a `ListView` control to the form. Set the following properties to get a similar look as in the figure above: **FullRowSelect** - True **GridLines** - True **View** - Details
4.	Add the columns for *IPAdress*, *ID, Availability, Virtual, System name, RobotWare Version* and *Controller name* and adjust the width of the columns.
5.	Add a `Panel` with a `Button` under the listview. Set the `Text` of the button.

Implement network scanning

To find all controllers on the network we start by declaring these member variables in the class `Form1`

VB:

```
Private scanner As NetworkScanner = Nothing
Private controller As Controller = Nothing
Private tasks As Task() = Nothing
Private networkWatcher As NetworkWatcher = Nothing
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
private NetworkScanner scanner = null;
private Controller controller = null;
private Task[] tasks = null;
private NetworkWatcher networkwatcher = null;
```

As the application is supposed to scan the network as soon as it is started, we can put the code for it in the `Form1_Load` event handler, like this:

VB:

```
Me.scanner = New NetworkScanner
Me.scanner.Scan()
Dim controllers As ControllerInfoCollection =
     Me.scanner.Controllers
Dim controllerInfo As ControllerInfo = Nothing
Dim item As ListViewItem
For Each controllerInfo In controllers
  item = New ListViewItem(controllerInfo.IPAddress.ToString())
  item.SubItems.Add(controllerInfo.Id)
  item.SubItems.Add(controllerInfo.Availability.ToString())
  item.SubItems.Add(controllerInfo.IsVirtual.ToString())
  item.SubItems.Add(controllerInfo.SystemName)
  item.SubItems.Add(controllerInfo.Version.ToString())
  item.SubItems.Add(controllerInfo.ControllerName)
  Me.listView1.Items.Add(item)
  item.Tag = controllerInfo
Next
```

C#:

```
this.scanner = new NetworkScanner();
this.scanner.Scan();
ControllerInfoCollection controllers = scanner.Controllers;
ListViewItem item = null;
foreach (ControllerInfo controllerInfo in controllers)
{
   item = new ListViewItem(controllerInfo.IPAddress.ToString());
   item.SubItems.Add(controllerInfo.Id);
   item.SubItems.Add(controllerInfo.Availability.ToString());
   item.SubItems.Add(controllerInfo.IsVirtual.ToString());
   item.SubItems.Add(controllerInfo.SystemName);
   item.SubItems.Add(controllerInfo.Version.ToString());
   item.SubItems.Add(controllerInfo.ControllerName);
   this.listView1.Items.Add(item);
   item.Tag = controllerInfo;
 }
```

© Copyright 2010 ABB. All rights reserved.

Add a network watcher

By implementing a `NetworkWatcher` the application can supervise the network and detect when controllers are lost or added. This example shows how to program network supervision, and how to add a detected controller to the listview.

After having added a `NetworkWatcher` object to the `FormLoad` event handler, we add a subscription to its `Found` event.

VB:

```
Me.networkWatcher = New NetworkWatcher(Me.scanner.Controllers)
AddHandler Me.networkWatcher.Found, AddressOf Me.HandleFoundEvent
AddHandler Me.networkWatcher.Lost, AddressOf Me.HandleLostEvent
Me.networkWatcher.EnableRaisingEvents = True
```

C#:

```
this.networkwatcher = new NetworkWatcher(scanner.Controllers);
            this.networkwatcher.Found += new
    EventHandler<NetworkWatcherEventArgs>(HandleFoundEvent);
            this.networkwatcher.Lost += new
    EventHandler<NetworkWatcherEventArgs>(HandleLostEvent);
            this.networkwatcher.EnableRaisingEvents = true;
```

NOTE!

In C# the event handler skeleton is auto generated using the Tab key twice after "+=" in the above statements. If you prefer, you can use a simplified syntax when using generic event handlers:

```
networkwatcher.Found += HandleFoundEvent;
```

Handle event

As the events will be received on a background thread and should result in an update of the user interface the `Invoke` method must be called in the event handler. For more information on how to force execution from background to GUI thread, see *Invoke method on page 45*.

VB:

```
Private Sub HandleFoundEvent(ByVal sender As Object, ByVal e As
     NetworkWatcherEventArgs)
  Me.Invoke(New EventHandler(Of
       NetworkWatcherEventArgs)(AddControllerToListView), New
       [Object]() {Me, e})
 End Sub
```

C#:

```
void HandleFoundEvent(object sender, NetworkWatcherEventArgs e)
{
  this.Invoke(new
       EventHandler<NetworkWatcherEventArgs>(AddControllerToList
       View), new Object[] { this, e });
}
```

© Copyright 2010 ABB. All rights reserved.

This event handler updates the user interface:

VB:

```
Private Sub AddControllerToListView(ByVal sender As Object, ByVal
     e As NetworkWatcherEventArgs)
  Dim controllerInfo As ControllerInfo = e.Controller
  Dim item As New
       ListViewItem(controllerInfo.IPAddress.ToString())
  item.SubItems.Add(controllerInfo.Id)
  item.SubItems.Add(controllerInfo.Availability.ToString())
  item.SubItems.Add(controllerInfo.IsVirtual.ToString())
  item.SubItems.Add(controllerInfo.SystemName)
  item.SubItems.Add(controllerInfo.Version.ToString())
  item.SubItems.Add(controllerInfo.ControllerName)
  Me.listView1.Items.Add(item)
  item.Tag = controllerInfo
End Sub
```

C#:

```
private void AddControllerToListView(object
     sender,NetworkWatcherEventArgs e)
{
  ControllerInfo controllerInfo = e.Controller;
  ListViewItem item = new
       ListViewItem(controllerInfo.IPAddress.ToString());
              item.SubItems.Add(controllerInfo.Id);
  item.SubItems.Add(controllerInfo.Availability.ToString());
  item.SubItems.Add(controllerInfo.IsVirtual.ToString());
  item.SubItems.Add(controllerInfo.SystemName);
  item.SubItems.Add(controllerInfo.Version.ToString());
  item.SubItems.Add(controllerInfo.ControllerName);
  this.listView1.Items.Add(item);
  item.Tag = controllerInfo;
 }
```

Establish connection to controller

When you double-clicks a controller in the list a connection to that controller should be established and you should be logged on. Use this procedure to implement the functionality:

Step	Action
1.	Generate the `DoubleClick` event of the `ListView`.
2.	In the event handler create a `Controller` object that represents the selected robot controller.
3.	Log on to the selected controller. For more information, see the code sample of *Implement event handler on page 57*.

Implement event handler

This example shows the code of the `ListView.DoubleClick` event handler:

`VB:`

© Copyright 2010 ABB. All rights reserved.

```
Dim item As ListViewItem = Me.listView1.SelectedItems(0)
If item.Tag IsNot Nothing Then
    Dim controllerInfo As ControllerInfo = DirectCast(item.Tag,
     ControllerInfo)
  If controllerInfo.Availability = Availability.Available Then
     If Me.controller IsNot Nothing Then
                 Me.controller.Logoff()
                 Me.controller.Dispose()
                 Me.controller = Nothing
     End If
  Me.controller = ControllerFactory.CreateFrom(controllerInfo)
          Me.controller.Logon(UserInfo.DefaultUser)
  Else

          MessageBox.Show("Selected controller not available.")
      End If
 End If

C#:
ListViewItem item = this.listView1.SelectedItems[0];
            if (item.Tag != null)
 {
  ControllerInfo controllerInfo = (ControllerInfo)item.Tag;
                 if (controllerInfo.Availability ==
       Availability.Available)
  {
  if (this.controller != null)
     {
                             this.controller.Logoff();
                             this.controller.Dispose();
                             this.controller = null;
                         }
  this.controller =
                    ControllerFactory.CreateFrom(controllerInfo);
                     this.controller.Logon(UserInfo.DefaultUser);
                 }
                 else

{
  MessageBox.Show("Selected controller not available.");
}
}
```

© Copyright 2010 ABB. All rights reserved.



NOTE!

The check to see whether the `Controller` object already exists is important, as you should explicitly log off and dispose of any existing controller object before creating a new one. The reason is that a logon session allocates resources that should not be kept longer than necessary.

Start program execution

The `Click` event handler of the **Start RAPID Program** button should start program execution of the first RAPID task.

Starting RAPID execution in manual mode can only be done from the FlexPendant, so we need to check that the controller is in automatic mode before trying. We then need to request mastership of `Rapid` and call the `Start` method. If mastership is already held, by ourselves or another client, an `InvalidOperationException` will be thrown. For further information, see *Mastership on page 33*.

It is necessary to release mastership whether or not the start operation succeeds. This can be done by calling `Release()` or `Dispose()` in a finally clause, as shown in the VB example, or by applying the `using` mechanism, as shown in the C# example.

VB:

```
Private Sub button1_Click(ByVal sender As Object, ByVal e As
     EventArgs)
   Try
     If controller.OperatingMode = ControllerOperatingMode.Auto
          Then
                tasks = controller.Rapid.GetTasks()
        Using m As Mastership =
             Mastership.Request(controller.Rapid)
        'Perform operation
         tasks(0).Start()
        End Using
     Else
        MessageBox.Show("Automatic mode is required to start
             execution from a remote client.")
     End If
        Catch ex As System.InvalidOperationException
        MessageBox.Show("Mastership is held by another client." &
             ex.Message)
        Catch ex As System.Exception
        MessageBox.Show("Unexpected error occurred: " & ex.Message)
  End Try
End Sub
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
private void button1_Click(object sender, EventArgs e)
  {
  try
     {
        if (controller.OperatingMode ==
              ControllerOperatingMode.Auto)
           {
              tasks = controller.Rapid.GetTasks();
              using (Mastership m
                    =Mastership.Request(controller.Rapid))
                 {
                 //Perform operation
                 tasks[0].Start();
                 }
           }
        else
           {
               MessageBox.Show("Automatic mode is required to start
                   execution from a remote client.");
           }
     }
    catch (System.InvalidOperationException ex)
     {
                          MessageBox.Show("Mastership is held by
          another client." + ex.Message);
     }
  catch (System.Exception ex)
     {
      MessageBox.Show("Unexpected error occurred: " + ex.Message);
                     }
}
```

© Copyright 2010 ABB. All rights reserved.

5.3. Discovery domain

Overview

To create a connection to the controller from a PC SDK application it has to make use of the Netscan functionality of the `Discovery` namespace. A `NetworkScanner` object must be created and a `Scan` call must be performed.

For the PC SDK to establish a connection either RobotStudio or Robot Communications Runtime must be installed on the PC hosting the PC SDK application. Robot Communications Runtime can be installed from C:\Program Files\ABB Industrial IT\Robotics IT\SDK\PC SDK 5.xx\Redistributable\RobotCommunicationRuntime, if RobotStudio is not installed.

To find out what controllers are available on the network you use the `NetworkScanner` methods `Scan`, `Find`, `GetControllers` and `GetRemoteControllers`.

NetworkScanner

The `NetworkScanner` class can be declared and represented at class level. No scanning is done until the `Scan` method is called. When the `GetControllers` method is called a collection of `ControllerInfo` objects is returned. Each such object holds information about a particular controller connected to the local network. Both virtual and real controllers are detected this way.

VB:

```
Private aScanner As NetworkScanner = New NetworkScanner
...' Somewhere in the code
aScanner.Scan()
Dim aCollection As ControllerInfo() = aScanner.GetControllers()
```

C#:

```
private NetworkScanner aScanner = new NetworkScanner();
... // Somewhere in the code
aScanner.Scan();
ControllerInfo[] aCollection = aScanner.GetControllers();
```

For a complete code sample, see *Implement network scanning on page 54*.

If only real controllers are of interest, you can first scan the network and then request only real controllers using the `NetworkScannerSearchCriterias` enumeration in the `GetControllers` method.

VB:

```
Dim aCollection As ControllerInfo() =
     aScanner.GetControllers(NetworkScannerSearchCriterias.Real)
```

C#:

```
ControllerInfo[] aCollection =
    aScanner.GetControllers(NetworkScannerSearchCriterias.Real);
```

If you know which controller system you want to connect to you can call the `Find` method, which finds a specified controller on the network. It takes the system ID as a System.Guid data type as argument. The system's globally unique identifier (GUID) can be find in the *system.guid* file in the INTERNAL folder of the robot system file system.

© Copyright 2010 ABB. All rights reserved.

ControllerInfo object

When a network scan is performed a collection of `ControllerInfo` objects is returned. The ControllerInfo object has information about availability. Remember that the ControllerInfo object is not updated when controller status changes. If you again need to find out if a controller is available, you need to perform a new network scan or use an existing `Controller` object and check the status directly.

Add controllers from outside local network

A network scan is done only on the local network. To detect controllers outside the local network you need to supply the IP address of the controller using the static `AddRemoteController` method or configuring it in the *App.config* file. For more information, see *PC application configuration on page 35*.

If you supply the controller IP address you either use a string argument or a `System.Net.IPAddress` object.

VB:

```
Dim ipAddress As System.Net.IPAddress
Try
   ipAddress = System.Net.IPAddress.Parse(Me.textBox1.Text)

      NetworkScanner.AddRemoteController(ipAddress)
  Catch ex As FormatException
      Me.textBox1.Text = "Wrong IP address format"
End Try
```

C#:

```
System.Net.IPAddress ipAddress;
try
  {

  ipAddress = System.Net.IPAddress.Parse(this.textBox1.Text);
                  NetworkScanner.AddRemoteController(ipAddress);


  }
catch (FormatException ex)
  {
  this.textBox1.Text = "Wrong IP address format";
              }
```

NetworkWatcher

By using a `NetworkWatcher` object you can supervise network changes and find out when a new controller is found or when a controller is lost. For a complete code example, see *Add a network watcher on page 56*.

© Copyright 2010 ABB. All rights reserved.

5.4. Accessing the controller

Controller object

By using a `Controller` object you can get access to the different domains of the robot controller, for example IO signals, RAPID, file system and elog messages.

To create a `Controller` object you normally make a call to the `ControllerFactory`:

VB:

```
Dim info As New ControllerInfo(New Guid("systemid"))
Dim aController As Controller
aController = ControllerFactory.CreateFrom(info)
```

C#:

```
ControllerInfo info = new ControllerInfo(new Guid("systemid"));
            Controller aController;
            aController = ControllerFactory.CreateFrom(info);
```

The argument is a `ControllerInfo` object, which may have been retrieved during a network scan, see *NetworkScanner on page 61*. It is also possible to add an optional argument if the IP address of the controller or the system ID (guid) should be used.

If the PC application is supposed to work with a single controller it can be specified in an *app.config* file. The default constructor can then be used to create the controller object, for example `aController = new Controller()`. For more information, see *<defaultSystem> on page 36*.

If several classes in your application need to access the controller, it is recommended that they all reference the same `Controller` object. This is done either by passing the `Controller` object as an argument to the constructor or by using a `Controller` property.



NOTE!

You should be aware that the .NET objects created for operations toward the robot controller will access native resources (C++ and COM code). The .NET garbage collector does not collect such objects, but these must be disposed of explicitly by the application programmer. For more information, see *Memory management in PC applications on page 63*.

Memory management in PC applications

An important feature of the .NET runtime environment is the garbage collector, which reclaims not referenced memory from the managed heap. Generally, this means that the programmer does not have to free memory that has been allocated by the use of `new`. There is no way of knowing exactly when garbage collection will be performed however.

For a PC application indeterministic deallocation of resources is usually not a problem (as opposed to a FlexPendant application, which runs on a small device with limited memory). The `IDisposable` interface, however, can be used in a PC application to obtain deterministic deallocation of resources. Using this interface you can make an explicit call to the `Dispose` method of any disposable object.

© Copyright 2010 ABB. All rights reserved.

If your application is running in a Single Threaded Apartment (STA) the `Dispose` call will dispose of managed objects, but native resources (created internally by the PC SDK) will remain. To release these native objects, the method `ReleaseUnmanagedResources` should be called periodically, for example when you clicks a certain button or each time data has been written to the controller. The method call is not costly.

For an application running in a Multi Threaded Apartment (MTA) the `Dispose` call will remove both managed and native objects.



NOTE!

The method `Controller.ReleaseUnmanagedResources` should be called once in a while to avoid memory leaks in PC SDK applications running in STA.

Dispose

It is the creator of a disposable object that is responsible for its lifetime and for calling `Dispose`. A check should be done that the object still exists and any subscriptions to controller events should be removed before the `Dispose` call. This is how you dispose of a `Controller` object:

VB:

```
If aController IsNot Nothing Then
  aController.Dispose()
  aController = Nothing
End If
```

C#:

```
if (aController != null)
{
  aController.Dispose();
  aController = null;
}
```

Logon and logoff

Before accessing a robot controller, the PC SDK application has to log on to the controller. The `UserInfo` parameter of the `Logon` method has a `DefaultUser` property that can be used. By default all robot systems have such a user configured.

VB:

```
aController.Logon(UserInfo.DefaultUser)
```

C#:

```
aController.Logon(UserInfo.DefaultUser);
```

If it is necessary for your application to handle users with different rights, these users can be configured by using the `UserAuthorizationManagement` namespace or by using the UAS administration tool in RobotStudio. This is how you create a new `UserInfo` object for login purposes.

VB:

```
Dim aUserInfo As New UserInfo("name", "password")
```

C#:

```
UserInfo aUserInfo = new UserInfo("name", "password");
```

© Copyright 2010 ABB. All rights reserved.

NOTE!

It is necessary to log off from the controller at application shut down at the latest.

VB: `AController.LogOff()`

C#: `aController.LogOff();`

Mastership

In order to get write access to some of the controller domains the application has to request mastership. The `Rapid` domain, that is, tasks, programs, modules, routines and variables that exist in the robot system, is one such domain. The `Configuration` domain is another.

For more information, see *Mastership on page 33*.

It is important to release mastership after a modification operation. One way of doing this is applying the `using` statement, which results in an automatic disposal of the `Mastership` object at the end of the block. Another possibility is releasing mastership in a `Finally` block, which is executed after the `Try` and `Catch` blocks. See how it can be coded in the examples of *Start program execution on page 59*.

Controller events

The `Controller` object provides several public events, which enable you to listen to operating mode changes, controller state changes, mastership changes and so on.

VB:

```
AddHandler AController.OperatingModeChanged, AddressOf
     OperatingModeChanged
AddHandler AController.StateChanged, AddressOf StateChanged
AddHandler AController.ConnectionChanged, AddressOf
     ConnectionChanged
```

C#:

```
AController.OperatingModeChanged += new
     EventHandler<OperatingModeChangeEventArgs>(OperatingModeChan
     ged);
AController.StateChanged += new
     EventHandler<StateChangedEventArgs>(StateChanged);
AController.ConnectionChanged += new
     EventHandler<ConnectionChangedEventArgs>(ConnectionChanged);
```

NOTE!

Controller events use their own threads. Carefully study *Controller events and threads on page 44* to avoid threading conflicts.

NOTE!

PC SDK 5.09 and onwards uses the generic event handling introduced by .NET Framework 2.0.

CAUTION!

Do not rely on receiving an initial event when setting up/activating a controller event. There is no guarantee an event will be triggered, so you had better read the initial state from the controller.

© Copyright 2010 ABB. All rights reserved.

Backup and Restore

Using the `Controller` object you can call the `Backup` method. The argument is a string describing the directory path on the controller where the backup should be stored. You can also restore a previously backed up system. This requires mastership of `Rapid` and `Configuration` and can only be done in Auto mode.

Backup sample

As the backup process is performed asynchronously you can add an event handler to receive a `BackupCompleted` event when the backup is completed. The backup directory should be created in the system backup directory, or else an exception will be thrown.

VB:

```
Dim backupDir As String = "(BACKUP)$" & backupDirName
AddHandler Me.aController.BackupCompleted, AddressOf
     aController_BackupCompleted
Me.aController.Backup(backupDir)
```

C#:

```
string backupDir = "(BACKUP)$" + backupDirName;
this.aController.BackupCompleted += new
     EventHandler<BackupEventArgs>(aController_BackupCompleted);
this.aController.Backup(backupDir);
```

Restore sample

The `Restore` method is synchronous, that is, execution will not continue until the restore operation is completed.

VB:

```
Dim restoreDir As String = "(BACKUP)$" & dirName
Using mc As Mastership =
     Mastership.Request(Me.aController.Configuration), mr As
     Mastership = Mastership.Request(Me.aController.Rapid)
Me.aController.Restore(restoreDir, RestoreIncludes.All,
     RestoreIgnores.All)
End Using
```

C#:

```
string restoreDir = "(BACKUP)$" + dirName;
using (Mastership mc =
     Mastership.Request(this.aController.Configuration), mr =
           Mastership.Request(this.aController.Rapid))
{
  this.aController.Restore(restoreDir, RestoreIncludes.All,
       RestoreIgnores.All);
}
```

NOTE!

You need to be logged in with required grants to perform the above functions.

© Copyright 2010 ABB. All rights reserved.

VirtualPanel

You can programmatically change the operating mode of the *virtual IRC5* using the `VirtualPanel` class and its `ChangeMode` method. This blocks the application thread until you manually accepts the mode change to Auto using the Virtual FlexPendant. An alternative to blocking the application thread eternally is to add a time-out and use a `try-catch` block to catch the `TimeoutException`.

VB:

```
Dim vp As VirtualPanel = VirtualPanel.Attach(aController)
Try
  'user need to acknowledge mode change on flexpendent
  vp.ChangeMode(ControllerOperatingMode.Auto, 5000)
  Catch ex As ABB.Robotics.TimeoutException
     Me.textBox1.Text = "Timeout occurred at change to auto"
End Try
vp.Dispose()
```

C#:

```
VirtualPanel vp = VirtualPanel.Attach(aController);
try
{
  //user need to acknowledge mode change on flexpendent
  vp.ChangeMode(ControllerOperatingMode.Auto, 5000);
}
catch (ABB.Robotics.TimeoutException ex)
{
  this.textBox1.Text = "Timeout occurred at change to auto";
}
vp.Dispose();
```

There are also the asynchronous method calls `BeginChangeOperatingMode` and `EndChangeOperatingMode`. It is important to use the second method in the callback since it returns the waiting thread to the thread-pool.

VB:

```
Dim vp As VirtualPanel = VirtualPanel.Attach(aController)
vp.BeginChangeOperatingMode(ControllerOperatingMode.Auto, New
     AsyncCallback(ChangedMode), vp)
```

C#:

```
VirtualPanel vp = VirtualPanel.Attach(aController);
vp.BeginChangeOperatingMode(ControllerOperatingMode.Auto, new
     AsyncCallback(ChangedMode), vp);
```

© Copyright 2010 ABB. All rights reserved.

The callback method must have the following signature and call the `EndChangeOperatingMode` as well as dispose the `VirtualPanel`.

VB:

```
Private Sub ChangedMode(ByVal iar As IAsyncResult)
  Dim vp As VirtualPanel = DirectCast(iar.AsyncState,
       VirtualPanel)
  vp.EndChangeOperatingMode(iar)
  vp.Dispose()
  ......
End Sub
```

C#:

```
private void ChangedMode(IAsyncResult iar)
{
     VirtualPanel vp = (VirtualPanel) iar.AsyncState;
     vp.EndChangeOperatingMode(iar);
     vp.Dispose();
     ....
  }
```

Learn more

This Application manual only covers some of the PC SDK functionality. To get the full potential of the PC SDK you should make use of the *PC SDK Reference* located in the PC SDK installation directory. For more information, see *PC SDK Reference on page 52*.

You can also learn a lot by becoming an active member of the *RobotStudio Community*. Its PC SDK *User Forum* should be your number one choice when you find yourself stuck with a coding issue you cannot solve on your own. For more information, see *RobotStudio Community on page 16*.

© Copyright 2010 ABB. All rights reserved.

5.5 Rapid domain

5.5.1. Working with RAPID data

Overview

The `RapidDomain` namespace enables access to RAPID data in the robot system. There are numerous PC SDK classes representing the different RAPID data types. There is also a `UserDefined` class used for referring to the RECORD structures in RAPID.

The `ValueChanged` event enables notification from the controller when persistent RAPID data has changed.

To speed up event notification from the controller there is a new functionality in PC SDK 5.10, which allows you to set up subscription priorities. This possibility applies to I/O signals and persistent RAPID data. This mechanism is further described in *Implementing high priority data subscriptions on page 75*.



NOTE!

To read RAPID data you need to log on to the controller. To modify RAPID data you must also request mastership of the `Rapid` domain.

Providing the path to the RAPID data

To read or write to RAPID data you must first create a `RapidData` object. The path to the declaration of the data in the controller is passed as argument. If you do not know the path, you need to search for the RAPID data by using the `SearchRapidSymbol` functionality.

Direct access

Direct access requires less memory and is faster, and is therefore recommended if you do not need to use the task and module objects afterwards.

The following example shows how to create a `RapidData` object that refers to the instance “reg1” in the USER module.

VB:

```
Dim rd As RapidData = aController.Rapid.GetRapidData("T_ROB1",
    "user", "reg1")
```

C#:

```
RapidData rd = aController.Rapid.GetRapidData("T_ROB1", "user",
    "reg1");
```

Hierarchical access

If you need the task and module objects hierarchical access can be more efficient. `GetRapidData` exists in the Rapid, Task and Module class.

VB:

```
rd = aController.Rapid.GetTask("T_ROB1").GetModule("user").
GetRapidData("reg1")
```

C#:

```
rd = aController.Rapid.GetTask("T_ROB1").GetModule("user").
    GetRapidData("reg1");
```

© Copyright 2010 ABB. All rights reserved.

Accessing data declared in a shared module

If your application is to be used with a MultiMove system (one controller and several motion tasks/robots), it may happen that the RAPID instance you need to access is declared in a *Shared* RAPID module. Such a module can be used by all tasks, T_ROB1, T_ROB2 and so on.

The following example shows how to create a `RapidData` object that refers to the instance “reg100”, which is declared in a shared module.

C#:

```
Task tRob1 = aController.Rapid.GetTask("T_ROB1");
if (tRob1 != null)
{
  RapidData rData = tRob1.GetRapidData("user", "reg1");
}
```



NOTE!

From RobotWare 5.12 onwards, even if the data is declared in a Shared **Hidden** module it **can** be accessed by the PC SDK.

Creating an object representing the RAPID data value

The `RapidData` object stores the path to the RAPID data. But this is not enough if you want to access its value (at least not if you want to *modify* it). To do that you need to create another object, which represents the *value* of the RAPID data.

In the `RapidDomain` namespace there are types representing the different RAPID data types. To create the object needed to represent the RAPID data value you use the `RapidData` property `Value` and cast it to the corresponding type, for example `Num`, `Bool` or `Tooldata`.

To access the value of a RAPID data of the RAPID data type `bool`:

VB:

```
'declare a variable of data type RapidDomain.Bool
Dim rapidBool As ABB.Robotics.Controllers.RapidDomain.Bool
Dim rd As ABB.Robotics.Controllers.RapidDomain.RapidData =
     aController.Rapid.GetRapidData("T_ROB1", "MainModule",
     "flag")
'test that data type is correct before cast
If TypeOf rd.Value Is ABB.Robotics.Controllers.RapidDomain.Bool
     Then
  rapidBool = DirectCast(rd.Value,
       ABB.Robotics.Controllers.RapidDomain.Bool)
  'assign the value of the RAPID data to a local variable
  Dim boolValue As Boolean = rapidBool.Value
End If
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
//declare a variable of data type RapidDomain.Bool
ABB.Robotics.Controllers.RapidDomain.Bool rapidBool;
ABB.Robotics.Controllers.RapidDomain.RapidData rd =
     aController.Rapid.GetRapidData("T_ROB1", "MainModule",
     "flag");
//test that data type is correct before cast
if (rd.Value is ABB.Robotics.Controllers.RapidDomain.Bool)
{
  rapidBool =
       (ABB.Robotics.Controllers.RapidDomain.Bool)rd.Value;
  //assign the value of the RAPID data to a local variable
  bool boolValue = rapidBool.Value;
}
```

If you want only to read this variable you can use the following technique instead of creating a RapidDomain.`Bool` object:

VB:

```
Dim b As Boolean = Convert.ToBoolean(rd.Value.ToString())
```

C#:

```
bool b = Convert.ToBoolean(rd.Value.ToString());
```

The .NET type `ToolData` (representing the RAPID data type `tooldata`) can be created like this:

VB:

```
Dim aTool As ToolData
If TypeOf rd.Value Is ToolData Then
  aTool = DirectCast(rd.Value, ToolData)
End If
```

C#:

```
ToolData aTool;
if (rd.Value is ToolData)
{
  aTool = (ToolData) rd.Value;
}
```

IRapidData.ToString method

All RapidDomain structures representing RAPID data types implement the `IRapidData` interface. It has a `ToString` method, which returns the value of the RAPID data in the form of a string. This is a simple example:

```
string bValue = rapidBool.ToString();
```

The string is formatted according to the principle described in *IRapidData.FillFromString method on page 72*.

© Copyright 2010 ABB. All rights reserved.

The following is an example of a complex data type. The `ToolData` `Tframe` property is of type `Pose`. Its `Trans` value is displayed in a label in the format [x, y, z].

VB:

```
Me.label1.Text = aTool.Tframe.Trans.ToString()
```

C#:

```
this.label1.Text = aTool.Tframe.Trans.ToString();
```

IRapidData.FillFromString method

The `IRapidData` interface also has a `FillFromString` method, which fills the object with a valid RAPID string representation. The method can always be used when you need to modify RAPID data. Using the method with the `RapidDomain.Bool` variable used earlier in the chapter will look like this:

```
rapidBool.FillFromString("True")
```

Using it for a `RapidDomain.Num` variable is similar:

```
rapidNum.FillFromString("10")
```

String format

The format is constructed recursively. The following example illustrate it.

Example:

The `RapidDomain.Pose` structure represents the RAPID data type `pose`, which describes how a coordinate system is displaced and rotated around another coordinate system.

```
public struct Pose : IRapidData
{
  public Pos trans;
  public Orient rot;
}
```

The following is an example in RAPID:

```
VAR pose frame1;
...
frame1.trans := [50, 0, 40];
frame1.rot := [1, 0, 0, 0];
```

The frame1 coordinate transformation is assigned a value that corresponds to a displacement in position where X=50 mm, Y=0 mm, and Z=40 mm. There is no rotation.

The `RapidDomain.Pose` structure consists of two struct variables called *trans* and *rot* of the data types `Pos` and `Orient`. `Pos` has three floats and `Orient` consists of four doubles. The `FillFromString` format for a `Pose` object is “[[1.0, 0.0, 0.0, 0.0][10.0, 20.0, 30.0]]”.

The example shows how to write a new value to a RAPID `pose` variable:

VB:

```
If TypeOf rd.Value Is Pose Then
  Dim rapidPose As Pose = DirectCast(rd.Value, Pose)
  rapidPose.FillFromString("[[1.0, 0.5, 0.0, 0.0][10, 15, 10]]")
  rd.Value = rapidPose
End If
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
if (rd.Value is Pose)
{
  Pose rapidPose = (Pose) rd.Value;
  rapidPose.FillFromString("[[1.0, 0.5, 0.0, 0.0][10, 15, 10]]");
  rd.Value = rapidPose;
}
```

NOTE!

The string format must be carefully observed. If the string argument has a wrong format, a `RapidDataFormatException` is thrown.

Writing to RAPID data

Writing to RAPID data is possible only by using the type cast `RapidData` value, to which the new value is assigned. To write the new value to the RAPID data in the controller, you must assign the .NET object to the `Value` property of the `RapidData` object. The following example uses the rapidBool object created in *Creating an object representing the RAPID data value on page 70*.

VB:

```
'Assign new value to .Net variable
rapidBool.Value = False
'Request mastership of Rapid before writing to the controller
Me.master = Mastership.Request(Me.aController.Rapid)
'Change: controller is repaced by aController
rd.Value = rapidBool
'Release mastership as soon as possible
Me.master.Dispose()
```

C#:

```
//Assign new value to .Net variable
rapidBool.Value = false;
//Request mastership of Rapid before writing to the controller
this.master = Mastership.Request(this.aController.Rapid);
//Change: controller is repaced by aController
rd.Value = rapidBool;
//Release mastership as soon as possible
this.master.Dispose();
```

For more information on how the controller handles write accesa, see *Mastership on page 33* and for another code example of implementing mastership in a PC SDK application, see *Start program execution on page 59*.

The preceding example is simple, as the value to change was a simple `bool`. Often, however, RAPID uses complex structures. By using the `FillFromString` method you can assign a new `Value` to any `RapidData` and write it to the controller.

© Copyright 2010 ABB. All rights reserved.

The string must be formatted according to the principle described in the `IRapidData.FillFromString` section. The following example shows how to write a new value to the `pos` structure (x, y, z) of a RAPID tooldata:

VB:

```
Dim aPos As New Pos()
aPos.FillFromString("[2,3,3]")
aTool.Tframe.Trans = aPos
Using Mastership.Request(aController.Rapid)
  rd.Value = aTool
End Using
```

C#:

```
Pos aPos = new Pos();
aPos.FillFromString("[2,3,3]");
aTool.Tframe.Trans = aPos;
using (Mastership.Request(aController.Rapid))
{
  rd.Value = aTool;
}
```



NOTE!

The new value is not written to the controller until the last statement is executed.

Letting the user know that RAPID data has changed

In order to be notified that RAPID data has changed you need to add a subscription to the `ValueChanged` event of the `RapidData` instance. However, that this only works for **persistent** RAPID data.

Add subscription

This is how you add a subscription to the `ValueChanged` event:

VB:

```
AddHandler rd.ValueChanged, AddressOf rd_ValueChanged
```

C#:

```
rd.ValueChanged += new
      EventHandler<DataValueChangedEventArgs>(rd_ValueChanged);
```

Handle event

The following example shows the implementation of the event handler. Remember that controller events use their own threads, and avoid Winforms threading problems by the use of `Control.Invoke`, which forces the execution from the background thread to the GUI thread.

VB:

```
Private Sub rd_ValueChanged(ByVal sender As Object, ByVal e As
      DataValueChangedEventArgs)
  Me.Invoke(New EventHandler(UpdateGUI), sender, e)
End Sub
```

© Copyright 2010 ABB. All rights reserved.

C#

```
private void rd_ValueChanged(object sender,
     DataValueChangedEventArgs e)
{
  this.Invoke(new EventHandler(UpdateGUI), sender, e);
}
```

To learn more about potential threading conflicts in PC SDK applications, see *Controller events and threads on page 44*.

Read new value from controlller

Update the user interface with the new value. As the value is not part of the event argument, you must use the `RapidData` `Value` property to retrieve the new value:

VB:

```
Private Sub UpdateGUI(ByVal sender As Object, ByVal e As
     System.EventArgs)
  Dim tool1 As ToolData = DirectCast(Me.rd.Value, ToolData)
  Me.label1.Text = tool1.Tframe.Trans.ToString()
End Sub
```

C#

```
private void UpdateGUI(object sender, System.EventArgs e)
{
  ToolData tool1 = (ToolData)this.rd.Value;
  this.label1.Text = tool1.Tframe.Trans.ToString();
}
```

NOTE!

Subscriptions work only for RAPID data declared as PERS.

Implementing high priority data subscriptions

To speed up event notification from the controller, it is possible to set up subscription priorities for persistent RAPID data. To do this, you can use the `Subscribe` method and the enumeration `EventPriority` as argument. The following example shows an ordinary signal subscription and a subscription with high priority:

VB:

```
AddHandler rd.ValueChanged, AddressOf rd_ValueChanged
rd.Subscribe(rd_ValueChanged, EventPriority.High)
```

C#:

```
 rd.ValueChanged += new
     EventHandler<DataValueChangedEventArgs>(rd_ValueChanged);
rd.Subscribe(rd_ValueChanged, EventPriority.High);
```

To deactivate subscriptions with high priority you can call the `Unsubscribe` method as described in the following example:

VB:

```
rd.Unsubscribe(rd_ValueChanged)
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
rd.Unsubscribe(rd_ValueChanged);
```

NOTE!

High priority subscriptions can be used for I/O signals and RAPID data declared PERS. The controller can handle 64 high priority subscriptions.

RapidData disposal

You are recommended to dispose the `RapidData` objects when they are no longer needed. For more information, see *Memory management in PC applications on page 63*.

VB:

```
If rd IsNot Nothing Then
  rd.Dispose()
  rd = Nothing
End If
```

C#:

```
 if (rd != null)
{
  rd.Dispose();
  rd = null;
}
```

© Copyright 2010 ABB. All rights reserved.

5.5.2. Handling arrays

Overview

In RAPID you can have up to three dimensional arrays. These are accessible by using a `RapidData` object like for any other RAPID data.

There are mainly two ways of accessing each individual element of an array: by indexers or by an enumerator.

ArrayData object

If the `RapidData` references a RAPID array is `Value` property returns an object of `ArrayData` type. Before making a cast, check the type using the `is` operator or by using the `IsArray` property on the `RapidData` object.

VB:

```
Dim rd As RapidData = aController.Rapid.GetRapidData("T_ROB1",
     "user", "num_array")
If rd.IsArray Then
  Dim ad As ArrayData = DirectCast(rd.Value, ArrayData)
  .....
End If
```

C#:

```
 RapidData rd = aController.Rapid.GetRapidData("T_ROB1", "user",
     "num_array");
if (rd.IsArray)
{
   ArrayData ad = (ArrayData)rd.Value;
  .....
}
```

Array dimensions

The dimension of the array is returned by the Rank property. If you need to check the length of the individual arrays you can use the `GetLength` method on the `ArrayData` object passing the dimension index as argument.

VB:

```
Dim aRank As Integer = ad.Rank
Dim len As Integer = ad.GetLength(aRank)
```

C#:

```
int aRank = ad.Rank;
int len = ad.GetLength(aRank);
```

Array item access by using indexers

By the use of indexers you can access each array element, even in three dimensional arrays. A combination of the `GetLength` method and `For` loops makes it possible to access any item:

VB:

```
Dim aSum As Double = 0R
```

© Copyright 2010 ABB. All rights reserved.

```
Dim aNum As Num
Dim rd As RapidData = aController.Rapid.GetRapidData("T_ROB1",
     "user", "num_array")
Dim ad As ArrayData = DirectCast(rd.Value, ArrayData)
Dim aRank As Integer = ad.Rank
If ad.Rank = 1 Then
    For i As Integer = 1 To ad.Length
        aNum = DirectCast(ad(i), Num)
        aSum += CDbl(aNum)
    Next
ElseIf ad.Rank = 2 Then
    For i As Integer = 1 To ad.GetLength(0)
        For j As Integer = 1 To ad.Length
            aNum = DirectCast(ad(i, j), Num)
            aSum += CDbl(aNum)
        Next
    Next
Else

    For i As Integer = 0 To ad.GetLength(0) - 1
        For j As Integer = 0 To ad.GetLength(1) - 1
            For k As Integer = 0 To ad.GetLength(2) - 1
                aNum = DirectCast(ad(i, j, k), Num)
                aSum += CDbl(aNum)
            Next
        Next
    Next
End If
```

C#:

```
 double aSum = 0d;
Num aNum;
RapidData rd = aController.Rapid.GetRapidData("T_ROB1", "user",
     "num_array");
ArrayData ad = (ArrayData)rd.Value;
int aRank = ad.Rank;
if (ad.Rank == 1)
{
for (int i = 1; i <= ad.Length; i++)
{
aNum = (Num)ad[i];
aSum += (double)aNum;
}
}
else if (ad.Rank == 2)
            {
                for (int i = 1; i <= ad.GetLength(0); i++)
```

© Copyright 2010 ABB. All rights reserved.

```
        {
            for (int j = 1; j <= ad.Length; j++)
            {
                aNum = (Num)ad[i, j];
                aSum += (double)aNum;
            }
        }
    }

    else
    {
        for (int i = 0; i < ad.GetLength(0); i++)
        {
            for (int j = 0; j < ad.GetLength(1); j++)
            {
                for (int k = 0; k < ad.GetLength(2); k++)
                {
                    aNum = (Num)ad[i, j, k];
                    aSum += (double)aNum;
                }
            }
        }
    }
```

Array item access using enumerator

You can also use the enumerator operation (`foreach`) like it is used by collections in .NET. Note that it can be used for both one dimension and multi-dimensional arrays to access each individual element. The previous example is a lot simpler this way:

VB:

```
Dim sum As Double = 0R
For Each ANum As Num In ad
    sum += CDbl(ANum)
Next
```

C#:

```
 double sum = 0d;
foreach (Num ANum in ad)
{
  sum += (double)ANum;
}
```

© Copyright 2010 ABB. All rights reserved.

5.5.3. ReadItem and WriteItem methods

Overview

An alternative way of accessing RAPID data stored in an array are the `ReadItem` and `WriteItem` methods.

ReadItem method

Using the `ReadItem` method you can directly access a `RapidData` item in an array, for example an array with *RobTargets* or *Nums*. The index to the item is explicitly specified in the `ReadItem` call. The first item is in position 1, that is, the array is 1-based as in RAPID.

VB:

```
Dim aNum As Num = DirectCast(rd.ReadItem(1, 2), Num)
```

C#:

```
Num aNum = (Num)rd.ReadItem(1, 2);
```

This example retrieves the second *Num* value in the first array of the RAPID data variable referenced by rd.

WriteItem method

It is possible to use the `WriteItem` method to write to an individual RAPID data item in an array. The following example shows how to write the result of an individual robot operation into an array representing a total robot program with several operations:

VB:

```
Dim aNum As New Num(OPERATION_OK)
rd.WriteItem(aNum, 1, 2)
```

C#:

```
Num aNum = new Num(OPERATION_OK);
rd.WriteItem(aNum, 1, 2);
```



NOTE!

If the index is not in the range specified, an `IndexOutOfRangeException` will be thrown.

© Copyright 2010 ABB. All rights reserved.

5.5.4. UserDefined data

Overview

RECORD structures are common in RAPID code. To handle these unique data types, a `UserDefined` class is available. This class has properties and methods to handle individual components of a RECORD.

In some cases implementing your own structure can improve application design and code maintenance.

Creating UserDefined object

The `UserDefined` constructor takes a `RapidDataType` object as argument. To retrieve a `RapidDataType` object you need to provide a `RapidSymbol` or the path to the declaration of the RAPID data type.

The following example creates a `UserDefined` object representing the RAPID RECORD *processdata*:

VB:

```
Dim rdt As RapidDataType
rdt = Me.aController.Rapid.GetRapidDataType("T_ROB1", "user",
      "processdata")
Dim processdata As New UserDefined(rdt)
```

C#

```
RapidDataType rdt;
rdt = this.aController.Rapid.GetRapidDataType("T_ROB1", "user",
      "processdata");
UserDefined processdata = new UserDefined(rdt);
```

Reading UserDefined data

`UserDefined` can be used to read any kind of RECORD variable from the controller. The individual components of the RECORD are accessible using the `Components` property and an index. Each `Component` can be read as a string.

VB:

```
Dim processdata As New UserDefined(rdt)
processdata = DirectCast(rd.Value, UserDefined)
Dim no1 As String = processdata.Components(0).ToString()
Dim no2 As String = processdata.Components(1).ToString()
```

C#:

```
UserDefined processdata = new UserDefined(rdt);
processdata = (UserDefined)rd.Value;
string no1 = processdata.Components[0].ToString();
string no2 = processdata.Components[1].ToString();
```

Each individual string can then be used in a `FillFromString` method to convert the component into a specific data type, for example `RobTarget` or `ToolData`. For more information, see *IRapidData.FillFromString method on page 72*.

© Copyright 2010 ABB. All rights reserved.

Writing to UserDefined data

If you want to modify `UserDefined` data and write it to the controller, you must first read the `UserDefined` object and the apply new values using the `FillFromString` method. Then you need to perform a write operation using the `RapidData.Value` property.

VB:

```
processdata.Components(0).FillFromString("[0,0,0]")
processdata.Components(1).FillFromString("10")
rd.Value = processdata
```

C#:

```
processdata.Components[0].FillFromString("[0,0,0]");
processdata.Components[1].FillFromString("10");
rd.Value = processdata;
```

For more information and code samples, see *IRapidData.FillFromString method on page 72* and *Writing to RAPID data on page 73*.

Recursively reading the structure of any RECORD data type

If you need to know the structure of a RECORD data type (built-in or user-defined) you must first retrieve the *record components* of the record. Then you need to iterate the record components and check if any of them are also records. This procedure must be repeated until all record components are atomic types.The following code example shows how to get information about the *robtarget* data type. The *robtarget* URL is “RAPID/robtarget” or just “robtarget”.

```
private void SearchRobtarget()
{
   RapidSymbolSearchProperties sProp =
       RapidSymbolSearchProperties.CreateDefault();
              sProp.Recursive = true;
              sProp.Types = SymbolTypes.Constant |
       SymbolTypes.Persistent;
              sProp.SearchMethod = SymbolSearchMethod.Block;

            RapidSymbol[] rsCol = tRob1.SearchRapidSymbol(sProp,
       "RAPID/robtarget", "p10");
              RapidDataType theDataType;
              if (rsCol.Length > 0)
              {
                  Console.WriteLine("RapidSymbol name = " +
       rsCol[0].Name);
                theDataType = RapidDataType.GetDataType(rsCol[0]);
              Console.WriteLine("DataType = " + theDataType.Name);
                  if (theDataType.IsRecord)
                  {
                      RapidSymbol[] syms =
       theDataType.GetComponents();
                      SearchSymbolStructure(syms);
                  }
```

© Copyright 2010 ABB. All rights reserved.

```
            }
}
private void SearchSymbolStructure(RapidSymbol[] rsCol)
{
   RapidDataType theDataType;
  foreach (RapidSymbol rs in rsCol)
  {
  Console.WriteLine("RapidSymbol name = " + rs.Name);
  theDataType = RapidDataType.GetDataType(rs);
  Console.WriteLine("DataType = " + theDataType.Name);
  if (theDataType.IsRecord)
                {
                    RapidSymbol[] syms =
      theDataType.GetComponents();
                    SearchSymbolStructure(syms);
                }
  }
}
```

© Copyright 2010 ABB. All rights reserved.

The code example above produces the following printout:

```
RapidSymbol name = p10
DataType = robtarget

RapidSymbol name = trans
DataType = pos
RapidSymbol name = x
DataType = num
RapidSymbol name = y
DataType = num
RapidSymbol name = z
DataType = num

RapidSymbol name = rot
DataType = orient
RapidSymbol name = q1
DataType = num
RapidSymbol name = q2
DataType = num
RapidSymbol name = q3
DataType = num
RapidSymbol name = q4
DataType = num

RapidSymbol name = robconf
DataType = confdata
RapidSymbol name = cf1
DataType = num
RapidSymbol name = cf4
DataType = num
RapidSymbol name = cf6
DataType = num
RapidSymbol name = cfx
DataType = num

RapidSymbol name = extax
DataType = extjoint
RapidSymbol name = eax_a
```

© Copyright 2010 ABB. All rights reserved.

DataType = num

RapidSymbol name = eax_b

DataType = num

RapidSymbol name = eax_c

DataType = num

RapidSymbol name = eax_d

DataType = num

RapidSymbol name = eax_e

DataType = num

RapidSymbol name = eax_f

DataType = num

Implement your own struct representing a RECORD

The following example shows how you can create your own .NET data type representing a RECORD in the controller instead of using the `UsefDefined` type.

Creating ProcessData type

VB:

```
Dim rdt As RapidDataType = ctr.Rapid.GetRapidDataType("T_ROB1",
     "MyModule", "processdata")
Dim pc As New ProcessData(rdt)
pc.FillFromString(rd.Value.ToString())
```

C#

```
 RapidDataType rdt = ctr.Rapid.GetRapidDataType("T_ROB1",
     "MyModule", "processdata");
ProcessData pc = new ProcessData(rdt);
pc.FillFromString(rd.Value.ToString());
```

Implementing ProcessData struct

The following example shows how the new data type `ProcessData` may be implemented. This is done by using a .NET struct and letting `ProcessData` wrap the `UserDefined` object.

The struct implementation should include a `FillFromString` and `ToString` method, that is, inherit the `IRapidData` interface. Any properties and methods may also be implemented.

VB:

```
Public Structure ProcessData
  Implements IRapidData
  Private data As UserDefined

  Public Sub New(ByVal rdt As RapidDataType)
     data = New UserDefined(rdt)
  End Sub

  Private Property IntData() As UserDefined
  Get
     Return data
```

© Copyright 2010 ABB. All rights reserved.

```
  End Get

  Set(ByVal Value As UserDefined)
     data = Value
  End Set
  End Property
  .....
End Structure
```

C#:

```
public struct ProcessData: IRapidData
{
  private UserDefined data;

  public ProcessData(RapidDataType rdt)
  {
     data = new UserDefined(rdt);
  }
  private UserDefined IntData
  {
     get { return data; }
     set { data = value; }
  }

  public int StepOne
  {
     get
     {
        int res =
             Convert.ToInt32(IntData.Components[0].ToString());
        return res;
     }
     set
     {
        IntData.Components[0] = new Num(value);
     }
  }
}
```

Implementing IRapidData methods

This piece of code shows how the two `IRapidData` methods `ToString` and `FillFromString` can be implemented.

VB:

```
Public Sub FillFromString(ByVal newValue As String) Implements
     ABB.Robotics.Controllers.RapidDomain.IRapidData.FillFromStr
     ing
  IntData.FillFromString(newValue)
```

© Copyright 2010 ABB. All rights reserved.

```
End Sub

Public Overrides Function ToString() As String Implements
     ABB.Robotics.Controllers.RapidDomain.IRapidData.ToString
  Return IntData.ToString()
End Function
```

C#:

```
public void FillFromString(string newValue)
{
  IntData.FillFromString(newValue);
}

public override string ToString()
{
  return IntData.ToString();
}
```

NOTE! The `ToString` method has to use the `Overrides` keyword in Visual Basic and the `override` keyword in C#.

Property implementation

Each item in the RECORD structure should have a corresponding property in the extended .NET data type. The get and set methods have to implement the conversion from/to controller data type to .NET data type.

VB:

```
Public Property Step() As Integer
  Get
     Dim res As Integer =
          Convert.ToInt32(IntData.Components(0).ToString())
     Return res
  End Get
  Set(ByVal Value As Integer)
     Dim tmp As Num = New Num
     tmp.FillFromNum(Value)
     IntData.Components(0) = tmp
  End Set
End Property
```

C#:

```
  public int Step
  {
     get
     {
        int res =
             Convert.ToInt32(IntData.Components[0].ToString());
        return res;
```

© Copyright 2010 ABB. All rights reserved.

```
        }
        set
        {
            Num tmp = new Num();
            tmp.FillFromNum(value);
            IntData.Components[0] = tmp;
        }
    }
```

© Copyright 2010 ABB. All rights reserved.

5.5.5. RAPID symbol search

Overview

Most RAPID elements (variables, modules, tasks, records and so on.) are members of a symbol table, in which their names are stored as part of a program tree structure.

It is possible to search this table and get a collection of `RapidSymbol` objects, each one including the RAPID object name, location, and type.

Search method

The search must be configured carefully, due to the large amount of RAPID symbols in a system. To define a query you need to consider from where in the program tree the search should be performed, which symbols are of interest, and what information you need for the symbols of interest. To enable search from different levels, the `SearchRapidSymbol` method is a member of several different SDK classes, for example `Task`, `Module`, and `Routine`. The following example shows a search performed with `Task` as the starting point:

VB:

```
Dim RSCol As RapidSymbol()
RSCol = ATask.SearchRapidSymbol(SProp, "num", string.Empty)
```

C#:

```
 RapidSymbol[] rsCol;
rsCol = aTask.SearchRapidSymbol(sProp, "num", string.Empty)
```

The `SearchRapidSymbol` method has three arguments. The first argument, of data type `RapidSymbolSearchProperties`, is detailed in the next section. The second and third arguments are detailed in the following sections.

Search properties

The `RapidSymbolSearchProperties` type is complex and requires some knowledge about RAPID concepts.

It is used to specify search method, type of RAPID symbol to search for, whether the search should be recursive, whether the symbols are local and/or global, and whether or not the search result should include only symbols currently used by a program. If a property is not valid for a particular symbol, it will be discarded and will not exclude the symbol from the search result.

The table describes the different properties of `RapidSymbolSearchProperties`.

Property	Description
SearchMethod	Specifies the direction of the search, which can be `Block` (down) or `Scope` (up). Example: If the starting point of the search is a routine, a block-search will return the symbols declared within the routine, whereas a scope-search will return the symbols accessible from the routine.
Types	Specifies which RAPID type(s) you want to search for. The `SymbolTypes` enumeration includes `Constant`, `Variable`, `Persistent`, `Function`, `Procedure`, `Trap`, `Module`, `Task`, `Routine`, `RapidData`. and so on. (`Routine` includes `Function`, `Procedure` and `Trap`. `RapidData` includes `Constant`, `Variable` and `Persistent`.)

© Copyright 2010 ABB. All rights reserved.

Property	Description
Recursive	For both block and scope search it is possible to choose if the search should stop at the next scope or block level or recursively continue until the root (or leaf) of the symbol table tree is reached.
GlobalSymbols	Specifies whether global symbols should be included.
LocalSymbols	Specifies whether local symbols should be included.
InUse	Specifies whether only symbols in use by the loaded RAPID program should be searched.

Default instance

`RapidSymbolSearchProperties` has several static methods that return a default instance.

VB:

```
Dim SProp As RapidSymbolSearchProperties =
     RapidSymbolSearchProperties.CreateDefault()
```

C#:

```
RapidSymbolSearchProperties sProp =
     RapidSymbolSearchProperties.CreateDefault();
```

The default instance has the following values:.

Property	Description
SearchMethod	SymbolSearchMethod.Block
Types	SymbolTypes.NoSymbol
Recursive	True
GlobalSymbols	True
LocalSymbols	True
InUse	True

Using this instance you can specify the search properties of the search you want to perform.

Example:

VB:

```
SProp.SearchMethod = SymbolSearchMethod.Scope
SProp.Types = SymbolTypes.Constant Or SymbolTypes.Persistent
SProp.Recursive = False
```

C#:

```
sProp.SearchMethod = SymbolSearchMethod.Scope;
sProp.Types = SymbolTypes.Constant | SymbolTypes.Persistent
sProp.Recursive = false;
```



NOTE!

The default instance has the property `Types` set to `NoSymbol`. It must be specified in order for a meaningful search to be performed!



NOTE!

The `Types` property allows you to combine several types in a search. See the preceding example.

© Copyright 2010 ABB. All rights reserved.

NOTE!

See *PC SDK Reference* for the static methods `CreateDefaultForData` and `CreateDefaultForRoutine`.

Data type argument

The second argument of the `SearchRapidSymbol` method is the RAPID data type written as a string. The data type should be written with small letters, for example “num”, “string” or “robtarget”. It can also be specified as `string.Empty`.

NOTE!

To search for a `UserDefined` data type, the complete path to the module that holds the RECORD definition must be passed. For example:

```
result = tRob1.SearchRapidSymbol(sProp,"RAPID/T_ROB1/MyModule/
MyDataType", string.Empty);
```

However, if `MyModule` is configured as -*Shared* the system sees its data types as installed, and the task or module should **not** be included in the path

```
result = tRob1.SearchRapidSymbol(sProp,"MyDataType", string.Empty);
```

Symbol name argument

The third argument is the name of the RAPID symbol. It can be specified as `string.Empty` if the name of the symbol to retrieve is not known, or if the purpose is to search ALL “num” data in the system.

Instead of the name of the RAPID symbol a regular expression can be used. The search mechanism will then match the pattern of the regular expression with the symbols in the symbol table. The regular expression string is not case sensitive.

A regular expression is a powerful mechanism. It may consist of ordinary characters and meta characters. A meta character is an operator used to represent one or several ordinary characters, and the purpose is to extend the search.

Within a regular expression, all alphanumeric characters match themselves, that is, the pattern “abc” will only match a symbol named “abc”. To match all symbol names containing the character sequence “abc”, it is necessary to add some meta characters. The regular expression for this is “.*abc.*”.

The available meta character set is shown below:

Expression	Meaning
.	Any single character
^	Any symbol starting with
[s]	Any single character in the non-empty set s, where s is a sequence of characters. Ranges may be specified as c-c.
[^s]	Any single 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 for separate that regular expression from another.

© Copyright 2010 ABB. All rights reserved.

Expression	Meaning
r | r’	The regular expressions r or r’.
.*	Any character sequence (zero, one or several characters).

Example 1

"^c.*"

Returns all symbols starting with c or C.

Example 2

"^reg[1-3]"

Returns reg1, Reg1, REG1, reg2, Reg2, REG2, reg3, Reg3 and REG3.

Example 3

"^c.*|^reg[1,2]"

Returns all symbols starting with c or C as well as reg1, Reg1, REG1, reg2, Reg2 and REG2.

SearchRapidSymbol example

This example searches for VAR, PERS or CONST num data in a task and its modules. The search is limited to globally declared symbols. By default the search method is `Block`, so it does not have to be set.

VB:

```
Dim sProp As RapidSymbolSearchProperties =
     RapidSymbolSearchProperties.CreateDefault()
sProp.Types = SymbolTypes.Data
sProp.LocalSymbols = False
Dim rsCol As RapidSymbol()
rsCol = aTask.SearchRapidSymbol(sProp, "num", String.Empty)
```

C#:

```
RapidSymbolSearchProperties sProp =
     RapidSymbolSearchProperties.CreateDefault();
sProp.Types = SymbolTypes.Data;
sProp.LocalSymbols = false;
RapidSymbol[] rsCol;
rsCol = aTask.SearchRapidSymbol(sProp, "num", string.Empty);
```

Search for UserDefined RAPID data - example

In this example a user defined RECORD data type (“mydata”) is declared in a module (“myModule”). Assuming that the end-user can declare and use data of this data type in any program module, the search method must be Block (default). A search for all “mydata” instances may look like this:

VB:

```
Dim sProp As RapidSymbolSearchProperties =
     RapidSymbolSearchProperties.CreateDefault()
sProp.Types = SymbolTypes.Data
Dim rsCol As RapidSymbol()
rsCol = aTask.SearchRapidSymbol(sProp, "RAPID/T_ROB1/MyModule/
     mydata", String.Empty)
```

© Copyright 2010 ABB. All rights reserved.

```
    rsCol = aTask.SearchRapidSymbol(sProp, "mydata", String.Empty)
```

C#:

```
     RapidSymbolSearchProperties sProp =
         RapidSymbolSearchProperties.CreateDefault();
    sProp.Types = SymbolTypes.Data;
    RapidSymbol[] rsCol;
    rsCol = aTask.SearchRapidSymbol(sProp, "RAPID/T_ROB1/MyModule/
         mydata", string.Empty);
    rsCol = aTask.SearchRapidSymbol(sProp, "mydata", string.Empty);
```



NOTE!

If *myModule* is configured as -*Shared* and all *myData* instances are declared in *myModule,* the search method must be set to `Scope` and the `SearchRapidSymbol` call should look like this:

```
rsCol = aTask.SearchRapidSymbol(sProp, "mydata", string.Empty);
```

© Copyright 2010 ABB. All rights reserved.

5.5.6. Working with RAPID modules and programs

Overview

Using the `Task` object it is possible to load and save individual modules and programs. You can also unload programs, as well as reset the program pointer and start program execution.

NOTE!

All these operations require mastership of the RAPID domain. For more information, see *Mastership on page 65*.

Load modules and programs

To load a module or program file, you need the path to the file on the controller. While the file is loaded into memory the `RapidLoadMode` enumeration argument, `Add` or `Replace,specifies whether or not it should` replace old modules or programs.

If the file extension is not a valid module (mod or sys) or program (pgf) extension an `ArgumentException` is thrown.

VB:

```
Try
   aTask.LoadProgramFromFile(aPrgFileName, RapidLoadMode.Replace)
    aTask.LoadModuleFromFile(aModFileName, RapidLoadMode.Add)
Catch ex As ArgumentException
    Return
End Try
```

C#:

```
try
{
  aTask.LoadProgramFromFile(aPrgFileName, RapidLoadMode.Replace);
  aTask.LoadModuleFromFile(aModFileName, RapidLoadMode.Add);
}
catch (ArgumentException ex)
{
  return;
}
```

NOTE!

All program files must reside in the file system of the controller and not locally on the PC. In order to load a program from the PC, you must first download it to the controller by using the `FileSystem.PutFile` method. For more information, see *Saving files on page 113*.

NOTE!

If the User Authorization System of the controller is used by the PC SDK application, it is required that the logged on user has the UAS grant UAS_RAPID_LOADPROGRAM to load and unload RAPID programs. For more information about which grants are necessary for a specific PC SDK method, see the *PC SDK Reference*.

© Copyright 2010 ABB. All rights reserved.

Save programs and modules

You can save programs using the `Task.SaveProgramToFile` method and a single module by using the `Module.SaveToFile` method.

To unload a program after it has been saved to file you can call `DeleteProgram()`.

VB:

```
Dim taskCol As Task() = aController.Rapid.GetTasks()
For Each atask As Task In taskCol
    atask.SaveProgramToFile(saveDir)
    atask.DeleteProgram()
Next
```

C#:

```
Task[] taskCol = aController.Rapid.GetTasks();
foreach (Task atask in taskCol)
{
atask.SaveProgramToFile(saveDir);
atask.DeleteProgram();
}
```

In this example a module is saved to a file:

VB:

```
Dim aModule As [Module] = aTask.GetModule("user")
aModule.SaveToFile(aFilePath)
```

C#

```
Module aModule = aTask.GetModule("user");
aModule.SaveToFile(aFilePath);
```

ResetProgramPointer method

Using `ResetProgramPointer` you can set the program pointer to the main entry point of the task.

VB:

```
aTask.ResetProgramPointer()
```

C#:

```
aTask.ResetProgramPointer();
```

Start program

Starting program execution in the robot controller can only be done in automatic operating mode. There are several overloaded `Start` methods to use, the simplest way to start RAPID execution of a controller task is:

VB:

```
aTask.Start()
```

C#:

```
aTask.Start();
```

© Copyright 2010 ABB. All rights reserved.



NOTE!

If your application uses the User Authorization System of the controller (see *User Authorization System on page 46*), you should also check whether the current user has the grant UAS_RAPID_EXECUTE before calling the `Start` method.

Execution change event

It is possible to subscribe to events that occur when a RAPID program starts and stops. It is done like this:

VB:

```
AddHandler aController.Rapid.ExecutionStatusChanged, AddressOf
    Rapid_ExecutionStatusChanged
```

C#

```
aController.Rapid.ExecutionStatusChanged += new
    EventHandler<ExecutionStatusChangedEventArgs>(Rapid_Executi
    onStatusChanged);
```

For more information on how to write the event handler that is needed to update the GUI due to a controller event, see *Avoiding threading conflicts on page 107*and *Letting the user know that RAPID data has changed on page 74*.

© Copyright 2010 ABB. All rights reserved.

5.5.7. Enable operator response to RAPID UI-instructions from a PC

Remote operator dialog box

PC SDK supports operator dialog box to be launched on a PC instead of the FlexPendant when RAPID UI- and TP-instructions are executed. In this chapter this feature is referred to as *Remote operator dialog*. It enables an operator to give the feedback required by the RAPID program from a PC instead of using the FlexPendant.



NOTE!

Remote operator dialog can only be used with RobotWare 5.12 and later.

Supported RAPID instructions

The following RAPID instructions are supported:

- UIAlphaEntry
- UIListView
- UIMessageBox
- UIMsgBox
- UINumEntry
- UINumTune
- TPErase
- TPReadFK
- TPReadNum
- TPWrite

UIInstructionType

The PC SDK `UIInstructionType` enumeration defines the different RAPID instructions listed above. For a description of each instruction type, see *PC SDK Reference*. The following is an example of such a description.

Example `UIInstructionType.UIAlphaEntry` :

Member	Description
UIAlphaEntry	The UIAlphaEntry (User Interaction Alpha Entry) is used to let an operator communicate with the robot system via RAPID, by enabling him to enter a string from the FlexPendant or from a PC SDK application. After the operator has entered the text, it is transferred back to the RAPID program by calling UIAlphaEntryEventArgs.SendAnswer.



TIP!

For complete information about the usage in RAPID refer to *RAPID Technical reference manual* (accessiable from RobotStudio).

© Copyright 2010 ABB. All rights reserved.

Increased flexibility

Making use of the *Remote operator dialog* feature, the end-user of the robot system can choose whether to use the FlexPendant or the PC SDK application to answer a RAPID UI- or TP-instruction.

The FlexPendant will always show the operator dialog the usual way. If the operator responds from the PC the message on the FlexPendant will disappear.



NOTE!

The dialog box of the PC SDK application should disappear if the operator chooses to respond from the FlexPendant. This is handled by the PC SDK programmer.

Basic approach

The basic procedure for implementing *Remote operator dialog* in a PC SDK application is shown below. The same approach is used internally by the FlexPendant when it launches its operator view.

Step	Action
1	Set up a subscription to `UIInstructionEvent`.
2	In the event handler check the `UIInstructionEventType` from the event arguments. If `Post` or `Send` create an operator dialog by using the information provided by the event arguments.
3	To transfer the response of the end-user to the RAPID program call the `SendAnswer` method of the specialized `UIInstructionEventArgs` object.
4	Remove any existing operator dialog if you get a `UIInstructionEvent` of `UIInstructionEventType.Abort`.



NOTE!

The controller events are always received on a background thread and you need to enforce execution to the GUI thread by the use of `Invoke` before launching the operator dialog. For more information, see *Controller events and threads on page 44*.

UIInstructionEvent

To be notified when a UI-instruction event has occurred in the controller, you need to set up a subscription to `UIInstructionEvent`. To do that you use the `UIInstruction` property of the `Rapid` class, like this:

```
Controller c = new Controller();
            c.Rapid.UIInstruction.UIInstructionEvent += new
      UIInstructionEventHandler(OnUIInstructionEvent);
```



TIP!

For a code example including an event handler see UIInstructionEvent in the *PC SDK Reference*.

© Copyright 2010 ABB. All rights reserved.

UIInstruction event arguments

To create the dialog in accordance with the arguments of the RAPID instruction and to transfer the response of the operator back to the executing RAPID program, you can use the information of the event arguments.

UIInstructionEventArgs

The `UIInstructionEventArgs` object holds information about which RAPID task and which UI- or TP-instruction triggered the event. The following picture shows all `UIInstructionEventArgs` members.



7.5.7_1

`UIInstructionEventArgs` is a base class of several specialized classes, one for each UI- and TP- instruction. The specialized class holds the additional information needed to create the operator dialog, so type casting the `UIInstructionEventArgs` object to the correct specialized type is necessary. To do that you first check the `InstructionType` property, which you can see in the preceding image.

© Copyright 2010 ABB. All rights reserved.

UIListViewEventArgs

As an example of a specialized type, the members of the `UIListViewEventArgs` class are shown below. The `Buttons` and `ListItems` properties are of course crucial for creating the operator dialog.



ABB Robotics IRC5 PC SDK | Contents | Index | Home

UIListViewEventArgs Members

UIListViewEventArgs Class | Public Methods | Public Properties

Collapse All

Class

UIListViewEventArgs Class

Public Methods

	Name	Description
	SendAnswer	Send selection to ListView.

Public Properties

	Name	Description
	BtnArray	User defined buttons stored in an array. Only one of parameter Buttons or BtnArray can be used at the same time.
	Buttons	Defines the buttons to be displayed.
	DefaultIndex	The default selection in the list, corresponds to the index of the item in the array specified in the parameter ListItems.
	EventMessage	Any additional text specified by instruction
	ExecutionLevel	Task execution level
	Header	Header text to be written at the top of the message box.
	Icon	Defines the icon to be displayed.
	Instruction	Name of the instruction e.g. TPWrite, UIMessageBox
	InstructionEventType	UI-Instructions are either sent with POST or SEND. An ABORT event is sent when a SEND instruction is aborted.
	InstructionType	UI instruction type.
	ListItems	An array with one or several list items to be displayed.
	StackUrl	URL to task or task stack
	TaskName	RAPID task name

7.5.7_2

© Copyright 2010 ABB. All rights reserved.

UIInstructionEventType

An important property in the picture above is `UIInstructionEventType`. It is inherited from the base class and comes with all UI- and TP- instruction events.

The following table shows the members of the `UIInstructionEventType` enumeration.

Member	Description
Undefined	Undefined. Should not occur.
Post	Post event type, for example TPWrite, TPErase. When the event is of this type RAPID expects no response from the operator.
Send	Send event type, for example TPReadNum, UIListView. When the event is of this type the running RAPID program expects feedback from the operator before execution continuous.
Abort	When the controller gets a response from a client (the FlexPendant or a PC SDK application) it sends an event of `Abort` type. This tells all subscribing clients that the UI-Instruction has been aborted, closed or confirmed by the operator. When you get an event of this type you should remove any open operator dialog.



NOTE!

If the robot system has several RAPID tasks, it is necessary to keep track of which operator dialog belongs to which task, and so on.

A RAPID task can handle only one pending `Send`, and it is not guaranteed that an `Abort` event will always follow a `Send` event. Therefore, if you receive a new `Send` event from the same task without a preceding `Abort`event, you should remove the existing dialog and display the new one.

SendAnswer method

To transfer the response of the end-user back to the RAPID program, you can call the `SendAnswer` method. See the image of the `UIListViewEventArgs` class above. `SendAnswer` is called with different arguments depending on the RAPID instruction.

For example, if it is a `UIAlphaEntry` instruction you can send the string that the operator has entered as argument. But if it is a `UIListView` instruction the `SendAnswer` method will look like this:

```
public void SendAnswer(int listItemIdx, UIButtonResult btnRes);
```



NOTE!

There is no mastership handling involved in using *Remote operator dialog*.

© Copyright 2010 ABB. All rights reserved.

5.6. IO system domain

Overview

A robot system uses input and output signals to control processes. Signals can be of digital, analog, or group signal type. Such IO signals are accessible using the SDK.

Signal changes in the robot system are often significant, and there are many scenarios where end-users of the system need notification of signal changes.

To speed up event notification from the controller, there is new functionality in PC SDK 5.10, which allows you to set up subscription priorities. This possibility applies to I/O signals and persistent RAPID data. This mechanism is further described in *Implementing high priority event subscription on page 106*.

Accessing signals

Accessing signals is done through the `Controller` object and its property `IOSystem`, which represents the IO signal space in the robot controller.

To access a signal you need the system name of the signal. The object that is returned from the `IOSystem.GetSignal` method is of type `Signal`.

VB:

```
Dim signal1 As Signal =
     aController.IOSystem.GetSignal("signal_name")
```

C#:

```
Signal signal1 = aController.IOSystem.GetSignal("signal_name");
```

The returned `Signal` object has to be typecast to digital, analog or group signal. This example shows a how a signal of type `DigitalSignal` is created:

VB:

```
Dim diSig As DigitalSignal = DirectCast(signal1, DigitalSignal)
```

C#:

```
DigitalSignal diSig = (DigitalSignal)signal1;
```

This example shows a how an `AnalogSignal` is created:

VB:

```
Dim aiSig As AnalogSignal = DirectCast(Signal1, AnalogSignal)
```

C#:

```
AnalogSignal aiSig = (AnalogSignal)signal2
```

This example shows a how a `GroupSignal` is created:

VB:

```
Dim giSig As GroupSignal = DirectCast(signal3, GroupSignal)
```

C#:

```
GroupSignal giSig = (GroupSignal)signal3;
```

© Copyright 2010 ABB. All rights reserved.

NOTE!

Remember to call the `Dispose` method of the signal when it should no longer be used.

Getting signals using SignalFilter

Instead of just getting one signal at a time you can get a signal collection using a signal filter. Some of the `SignalFilter` flags are mutually exclusive, for example `SignalFilter.Analog` and `SignalFilter.Digital`. Others are inclusive, for example `SignalFilter.Digital` and `SignalFilter.Input`. You can combine the filter flags using the "|" character in C# and the `Or` operator in VB:

VB:

```
Dim aSigFilter As IOFilterTypes = IOFilterTypes.Digital Or
     IOFilterTypes.Input
Dim signals As SignalCollection =
     aController.IOSystem.GetSignals(aSigFilter)
```

C#:

```
 IOFilterTypes aSigFilter = IOFilterTypes.Digital |
     IOFilterTypes.Input;
SignalCollection signals =
     aController.IOSystem.GetSignals(aSigFilter);
```

The following code iterates the signal collection and adds all signals to a `ListView` control. The list has three columns displaying signal name, type, and value:

VB:

```
For Each signal As Signal In signals
    item = New ListViewItem(signal.Name)
    item.SubItems.Add(signal.Type.ToString())
    item.SubItems.Add(signal.Value.ToString())
    listView1.Items.Add(item)
Next
```

C#:

```
 foreach (Signal signal in signals)
{
item = new ListViewItem(signal.Name);
item.SubItems.Add(signal.Type.ToString());
item.SubItems.Add(signal.Value.ToString());
listView1.Items.Add(item);
}
```

If the signal objects are no longer needed they should be disposed of:

VB:

```
For Each signal As Signal In signals
    signal.Dispose()
Next
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
foreach (Signal signal in signals)
            {
                signal.Dispose();
            }
```

Reading IO signal values

The following examples show how to read a digital and an analog signal.

Digital signal

The following code reads the digital signal DO1 and selects a checkbox if the signal value is 1 (ON):

VB:

```
Dim sig As Signal = aController.IOSystem.GetSignal("DO1")
Dim digitalSig As DigitalSignal = DirectCast(sig, DigitalSignal)
Dim val As Integer = digitalSig.[Get]()
If val = 1 Then
    Me.checkBox1.Checked = True
End If
```

C#:

```
Signal sig = aController.IOSystem.GetSignal("DO1");
DigitalSignal digitalSig = (DigitalSignal)sig;
int val = digitalSig.Get();
if (val == 1)
{
this.checkBox1.Checked = true;
}
```

Analog signal

The following of code reads the value of the analog signal AO1 and displays it in a textbox:

VB:

```
Dim asig As Signal = aController.IOSystem.GetSignal("AO1")
Dim analogSig As AnalogSigna = DirectCast(asig, AnalogSignal)
Dim analogSigVal As Single = analogSig.Value
Me.textBox1.Text = analogSigVal.ToString()
```

C#:

```
Signal asig = aController.IOSystem.GetSignal("AO1");
AnalogSigna analogSig = (AnalogSignal)asig;
float analogSigVal = analogSig.Value;
this.textBox1.Text = analogSigVal.ToString();
```

© Copyright 2010 ABB. All rights reserved.

Writing IO signal values

The following section shows how the value of a digital or an analog IO signal can be modified by a PC SDK application.



NOTE!

In manual mode, a signal value can be modified only if the *Access Level* of the signal is *ALL*. If not, the controller has to be in auto mode.

Digital signal

The following code changes the value of a digital signal in the controller when you select/ unselect a checkbox:

VB:

```
Private Sub checkBox1_Click(ByVal sender As Object, ByVal e As
     EventArgs)
    If Me.checkBox1.Checked Then
        digitalSig.[Set]()
    Else
        digitalSig.Reset()
    End If
End Sub
```

C#:

```
private void checkBox1_Click(object sender, EventArgs e)
{
  if (this.checkBox1.Checked)
  {
     digitalSig.Set();
  }
  else
  {
     digitalSig.Reset();
  }
}
```

NOTE! You can also set the value using the `Value` property.

Analog signal

The following code writes the value entered in a text box to the analog signal AO1. The value is converted from string to a float before it is written to the controller:

VB:

```
Dim analogSigVal As Single = Convert.ToSingle(Me.textBox1.Text)
analogSig.Value = analogSigVal
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
float analogSigVal = Convert.ToSingle(this.textBox1.Text);
analogSig.Value = analogSigVal;
```

Listening to signal changes

Once a `Signal` object is available it is possible to add a subscription to its Changed event, which is triggered at a signal change such as changed value, changed simulated status or changed signal quality.

Visual Basic

```
Friend WithEvents sig As AnalogSignal
...
AddHandler sig.Changed, AddressOf sig_Changed
...
Private Sub sig_Changed(ByVal sender As Object, ByVal e As
     SignalChangedEventArgs)
.....
End Sub
```

C#

```
 sig.Changed += new
     EventHandler<SignalChangedEventArgs>(sig_Changed);
...
private void sig_Changed(object sender, SignalChangedEventArgs e)
{..... }
```

Start and stop subscriptions

It is recommended that you activate and deactivate subscriptions to the `Changed` event if these are not necessary throughout the lifetime of the application:

VB:

```
AddHandler sig.Changed, AddressOf sig_Changed
RemoveHandler sig.Changed, AddressOf sig_Changed
```

C#:

```
 sig.Changed += new
     EventHandler<SignalChangedEventArgs>(sig_Changed);
            sig.Changed -= new
     EventHandler<SignalChangedEventArgs>(sig_Changed);
```

Implementing high priority event subscription

To speed up event notification from the controller, it is possible to set up subscription priorities for I/O signals. To do this, you can use the `Subscribe` method and the enumeration `EventPriority` as argument. The example shows an ordinary signal subscription and a subscription with high priority:

VB:

```
AddHandler signal.Changed, AddressOf sig_Changed
signal.Subscribe(sig_Changed, EventPriority.High)
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
signal.Changed += new
     EventHandler<SignalChangedEventArgs>(sig_Changed);
            signal.Subscribe(sig_Changed, EventPriority.High);
```

To deactivate subscriptions with high priority you call the `Unsubscribe` method like this:

VB:

```
signal.Unsubscribe(sig_Changed)
```

C#:

```
signal.Unsubscribe(sig_Changed);
```

Limitations for high priority events

High priority subscriptions can be used for I/O signals and RAPID data declared PERS. The controller can handle 64 high priority subscriptions.

Avoiding threading conflicts

The controller events use their own threads, which are different from the application GUI thread. This can cause problems if you want to display signal changes in the application GUI. For more information, see *Controller events and threads on page 44*.

If an update of the user interface is not necessary, you do not need to take any special action, but can execute the event handler on the event thread. If, however, you need to show to the user that the signal has changed you should use the `Invoke` method. It forces execution to the window control thread and thus provides a solution to potential threading conflicts.

VB:

```
Me.Invoke(New EventHandler(Of SignalChangedEventArgs)(UpdateUI),
     New [Object]() {sender, e})
```

C#:

```
this.Invoke(new EventHandler<SignalChangedEventArgs>(UpdateUI),
     new Object[] { sender, e });
```

Reading the new value

The `SignalChangedEventArgs` object has a `NewSignalState` property, which has information about signal value, signal quality and whether the signal is simulated or not:

VB:

```
Private Sub UpdateUI(ByVal Sender As Object, ByVal e As
     SignalChangedEventArgs)
  Dim state As SignalState = e.NewSignalState
  Dim val As Single
  Val = state.Value
  Me.textBox1.Text = val.ToString()
  ....
End Sub
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
private void UpdateUI(object sender, SignalChangeEventArgs e)
{
  SignalState state = e.NewSignalState;
  ....
  float val = state.Value
  this.textBox1.Text = val.ToString()
}
```

NOTE!

There is no guarantee you will receive an initial event when setting up the subscription. To get initial information about the value of a signal, you should read it using the `Value` property.

NOTE!

Make sure the subscription is removed before you dispose of the signal. For more information, see *Memory management in PC applications on page 63*.

© Copyright 2010 ABB. All rights reserved.

5.7. Event log domain

Overview

Event log messages may contain information about controller status, RAPID execution, the running processes of the controller, and so on.

Using the SDK it is possible to either read messages in the queue or to use an event handler that will receive a copy of each new log message. An event log message contains queue type, event type, event time, event title, and message.

Accessing the controller event log

You can access the event log domain through the `Controller` property `EventLog`.

VB:

```
Private log As EventLog = aController.EventLog
```

C#:

```
private EventLog log = aController.EventLog;
```

Accessing event log categories

All event log messages are organized into categories. To search for an individual message you have to know what category it belongs to. The enumeration type, `CategoryType`, defines all available categories. You can get a category either by using the method `GetCategory` or by using the `Categories` property, which is an array of all available categories.

VB:

```
Dim cat As EventLogCategory
cat = Log.GetCategory(CategoryType.Program)
```

or

```
cat = log.Categories(4)
```

C#:

```
EventLogCategory cat;
cat = log.GetCategory(CategoryType.Program);
```

or

```
cat = log.GetCategory[4];
```



NOTE!

The `EventLogCategory` should be disposed of when it is no longer used.

Accessing event log messages

To access a message you use the `Messages` property of the `Category` object. A collection of messages is returned. The collection implements the `ICollection` and `IEnumerable` interfaces , which means you can use the common operations for collections. Access is done either using an index or by iterating using `foreach`.

VB:

```
Dim msg As EventLogMessage = cat.Messages(1)
```

© Copyright 2010 ABB. All rights reserved.

or

```
Dim msg As EventLogMessage
For Each msg In cat.Messages
  Me.textBox1.Text = msg.Title
  .....
Next Item
```

C#:

```
EventLogMessage msg = cat.Messages[1];
```

or

```
        foreach (EventLogMessage emsg in cat.Messages)

{
this.textBox1.Text = emsg.Title;
.....
        }
```

MessageWritten event

It is possible to add an event handler that is notified when a new messages is written to the controller event log. This is done by subscribing to the EventLog event `MessageWritten`.

The event argument is of type `MessageWrittenEventArgs` and has a `Message` property, which holds the latest event log message.

VB:

```
Private Sub log_MessageWritten(ByVal sender As Object, ByVal e As
     MessageWrittenEventArgs)
    Dim msg As EventLogMessage = e.Message
End Sub
```

C#:

```
private void log_MessageWritten(object sender,
     MessageWrittenEventArgs e)
{
  EventLogMessage msg = e.Message;
}
```



NOTE!

If the application user interface needs to be updated as a result of the event, you must delegate this job to the GUI thread using the `Invoke` method. For more information and code samples, see *Invoke method on page 45*.



TIP!

Find out more about the `EventLogDomain` in the PC SDK Reference help.

© Copyright 2010 ABB. All rights reserved.

5.8. Motion domain

Overview

The `MotionDomain` namespace lets you access the mechanical units of the robot system.

Motion system

You can access the motion system by using the `Controller` property `MotionSystem`.

VB:

```
Private aMotionSystem As MotionSystem
aMotionSystem = aController.MotionSystem
```

C#

```
private MotionSystem aMotionSystem;
aMotionSystem = aController.MotionSystem;
```

By using the `MotionSystem` object you can, for example, use its `SpeedRatio` property to find out about the current speed of the robot.

Accessing Mechanical units

The mechanical units can be of different types, for example a robot with a TCP, a multiple axes manipulator, or a single axis unit. All these are available through the MotionSystem property `MechanicalUnits`. If only the active mechanical unit is of interest you may use the `ActiveMechanicalUnit` property.

VB:

```
Dim aMechCol As MechanicalUnitCollection =
     aController.MotionSystem.MechanicalUnits
Dim aMechUnit As MechanicalUnit =
     aController.MotionSystem.ActiveMechanicalUnit
```

C#:

```
MechanicalUnitCollection aMechCol =
     aController.MotionSystem.MechanicalUnits;
MechanicalUnit aMechUnit =
     aController.MotionSystem.ActiveMechanicalUnit;
```

Mechanical unit properties and methods

There are numerous properties available for the mechanical unit, for example `Name`, `Model`, `NumberOfAxes`, `SerialNumber`, `CoordinateSystem`, `MotionMode`, `IsCalibrated`, `Tool` and `WorkObject`, and so on. It is also possible to get the current position of a mechanical unit as a `RobTarget` or `JointTarget`.

VB:

```
Dim aRobTarget As RobTarget =
     aController.MotionSystem.ActiveMechanicalUnit.GetPosition(C
     oordinateSystemType.World)
Dim aJointTarget As JointTarget =
     aController.MotionSystem.ActiveMechanicalUnit.GetPosition()
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
RobTarget aRobTarget =
     aController.MotionSystem.ActiveMechanicalUnit.GetPosition(C
     oordinateSystemType.World);
 JointTarget aJointTarget=
     aController.MotionSystem.ActiveMechanicalUnit.
GetPosition();
```



TIP!

To read Rapid data that is, RobTarget and JointTarget, user must Logon to controller

Find out more about the `MotionDomain` in the PC SDK Reference help.

Calibrating Axes

The revolution counters of a TCP mechanical unit is updated using the method `MechanicalUnit.SetRevolutionCounter`. For example, when the control system is disconnected and the robot axis is moved update the revolution counters.

`MechanicalUnit.GetMechanicalUnitStatus` property shows whether calibration is needed or not.

Fine calibration is performed using the method `MechanicalUnit.FineCalibrate`.

For more information, see *Reference Manual PC SDK*.

`FineCalibrate` and `SetRevolutionCounter` require the system to be either in 'Auto mode and Motors off state' or in 'manual mode and granted access' by the FlexPendant. `FineCalibrate` and `SetRevolutionCounter` also require the corresponding UAS grants.

For more information on how to perform calibration, see *Operating manual - IRC5 with FlexPendant*.



CAUTION!

If a revolution counter is incorrectly updated, it will cause incorrect robot positioning, which in turn may cause damage or injury.

© Copyright 2010 ABB. All rights reserved.

5.9. File system domain

Overview

Using the SDK it is possible to create, save, load, rename, and delete files in the controller file system. It is also possible to create and delete directories.

Accessing files and directories

You can access the file system domain through the `Controller` property `FileSystem`.

VB:

```
Dim aFileSystem As FileSystem = aController.FileSystem
```

C#:

```
FileSystem aFileSystem = aController.FileSystem;
```

Controller and PC directory

You can get and set the directory on the controller and on the local PC system using the `RemoteDirectory` and `LocalDirectory` properties.

VB:

```
Dim remoteDir As String = aController.FileSystem.RemoteDirectory
Dim localDir As String = aController.FileSystem.LocalDirectory
```

C#:

```
string remoteDir = aController.FileSystem.RemoteDirectory;
string localDir = aController.FileSystem.LocalDirectory;
```

Environment variables

When specifying file system paths you can use environment variables to denote the HOME, system, backup, and temp directories of the currently used system. When an application uses “(BACKUP)$” it is internally interpreted as the path to the backup directory of the current system. The other environment variables are: HOME, TEMP and SYSTEM.

Loading files

You can load a file from the controller to the PC using the `GetFile` method. The method generates an exception if the operation did not work. The arguments are complete paths including filenames.

VB:

```
aController.FileSystem.GetFile(remoteFilePath, localFilePath)
```

C#:

```
aController.FileSystem.GetFile(remoteFilePath, localFilePath);
```

Saving files

You can save a file on the controller file system by using the `PutFile` method. The method generates an exception if the operation did not work. The arguments are complete paths including filenames.

VB:

```
aController.FileSystem.PutFile(localFilePath, remoteFilePath)
```

© Copyright 2010 ABB. All rights reserved.

C#:

```
aController.FileSystem.PutFile(localFilePath, remoteFilePath);
```

CopyFile and CopyDirectory

The `PutFile` / `GetFile` methods generate a copy of a file and transfer it to or from the controller file system. Using the `CopyFile` and `CopyDirectory` you can create a copy directly on the controller:

VB:

```
aController.FileSystem.CopyFile(fromFilePath, toFilePath, True)
aController.FileSystem.CopyDirectory(fromDirPath, toDirPath,
     True)
```

C#:

```
aController.FileSystem.CopyFile(fromFilePath, toFilePath,true);
aController.FileSystem.CopyDirectory(fromDirPath,
     toDirPath,true);
```

Getting multiple files and directories

The `FileSystem` class has a method called `GetFilesAndDirectories`. It can be used to retrieve an array of `ControllerFileSystemInfo` objects with information about individual files and directories. The `ControllerFileSystemInfo` object can then be cast to either a `ControllerFileInfo` object or a `ControllerDirectoryInfo` object.

This example uses search pattern to limit the search.

VB:

```
Dim anArray As ControllerFileSystemInfo()
Dim info As ControllerFileSystemInfo
anArray = aController.FileSystem.GetFilesAndDirectories("search
     pattern")
For i As Integer = 0 To anArray.Length - 1
info = anArray(i)
.....
Next
```

C#:

```
ControllerFileSystemInfo[] anArray;
ControllerFileSystemInfo info;
anArray = aController.FileSystem.GetFilesAndDirectories("search
     pattern");
for (int i=0;i<anArray.Length;i++) {
  info = anArray[i];
  ......
}
```

© Copyright 2010 ABB. All rights reserved.

Using search patterns

As seen in the preceding example, you can use search patterns to locate files and directories using the `GetFilesAndDirectories` method. The matching process follows the Wildcard pattern matching in Visual Studio. The following is a brief summary:

Character in pattern	Matches in string
?	Any single character
*	Zero or more characters
#	Any single digit (0–9)
[*charlist*]	Any single character in *charlist*
[!*charlist*]	Any single character not in *charlist*



TIP!

Find out more about the `FileSystemDomain` in the PC SDK Reference help.

© Copyright 2010 ABB. All rights reserved.

5.10. Messaging domain

Overview

The `Messaging` domain of the PC SDK can be used to send and receive data between a PC SDK application and a RAPID task.

The corresponding RAPID functionality, *RAPID Message Queue*, includes RAPID data types and RAPID instructions and functions for sending and receiving data. It enables communication between RAPID tasks or between a RAPID task and a PC SDK application.

This section provides information about how to implement messaging in a PC SDK application. To make it work it is necessary to do part of the implementation in RAPID. In order to show how this can be done, a code example in C# and RAPID is provided at the end of the section.

NOTE!

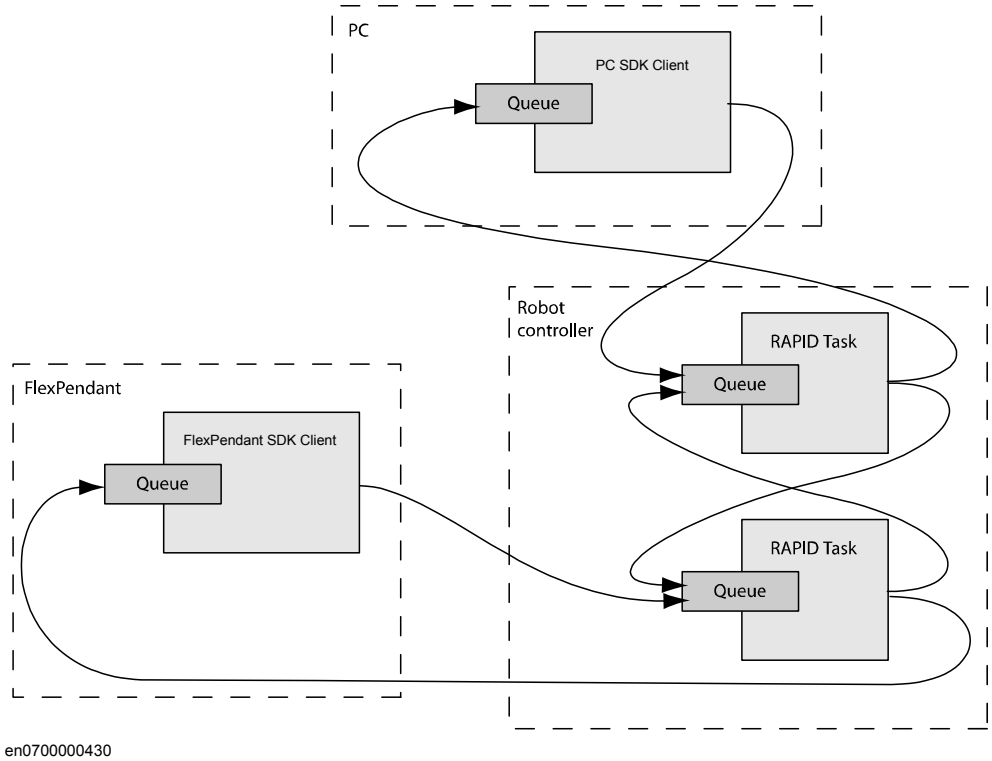
For more information on how to implement messaging in RAPID, see *Application manual - Robot communication and I/O Control*.

RobotWare option

The functionality in RAPID that is needed to utilize messaging - *RAPID Message Queue* - is included in the RobotWare options *PC Interface*, *FlexPendant Interface* and *Multitasking*. As *PC Interface* is required on a robot controller to be used with a PC SDK client, this means no extra option is needed to start using *RAPID Message Queue* with a PC SDK application.

Messaging illustration

The following illustration shows possible senders and receivers in the robot system. The arrows represent ways to communicate by posting a message to a queue.



en0700000430

© Copyright 2010 ABB. All rights reserved.

NOTE!

In principle, messages might as well be sent between a PC SDK client and a PC SDK application running on the FlexPendant.

NOTE!

The messaging functionality of the FlexPendant SDK has not yet been made public, but in PC SDK 5.11 a mechanism has been implemented, which allows advanced users to access it. You should contact the customer support if you need information about how to use it.

Benefits

Together with *RAPID Message Queue* the functionality of the `Messaging` domain represent a new, flexible way for a PC SDK application to interact with a RAPID task.

Messaging is usually done when a RAPID task is executing, but it is also possible to send a message to a RAPID task when it has been stopped. The RAPID interrupt will then occur once the RAPID task has been started.

An simple example of usage would be to set a flag from a PC SDK application in order to control the program flow in the RAPID program.

NOTE!

Sending messages can be done in both manual and auto mode. As opposed to using `RapidData` to modify a RAPID variable no mastership is required.

The Messaging namespace

The Microsoft Windows operating system provides mechanisms for facilitating communications and data sharing between applications. Collectively, activities enabled by these mechanisms are called *Interprocess communications* (IPC).

These are the classes and enumerations available in the `Messaging` namespace:

© Copyright 2010 ABB. All rights reserved.



7.9_1

The `Ipc` class is used to handle message queues with methods like `GetQueue`, `CreateQueue`, `DeleteQueue` and so on. When you have an `IpcQueue` object you can use its `Send` method to send an `IpcMessage` to a RAPID task or its `Receive` method to receive a message from a RAPID task.

When sending a message you use an existing queue in the controller as the `IpcQueue` object. The naming principle of queues in the controller is using the name of the corresponding task prefixed with “RMQ_”, e.g “RMQ_T_ROB1”. To receive a message from RAPID you must first create your own message queue and use that object with the `Receive` method.

NOTE!

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

© Copyright 2010 ABB. All rights reserved.

Basic approach

To utilize messaging in a PC SDK application, you need to do the implementation both in RAPID and in the PC application.

To send data from a PC application and receive it in a RAPID task:

1. In the PC application connect to the queue of the RAPID task.
2. Create the message.
3. Send the message.
4. In the RAPID program set up a trap routine that reads the message. Connect an interrupt so that the trap routine is called each time a new message appears.

For a complete code example using this scenario, see *Code example on page 122*.

What can be sent in a message?

In RAPID there is a `rmqmessage data type`. In the PC SDK the corresponding type is `IpcMessage`. An `IpcMessage` object stores the actual data in the message, but also information about message size, who the sender is and so on.

The data in a message is a pretty-printed string with data type name (and array dimensions) followed by the actual data value. The data type can be any RAPID data type. Arrays and user defined records are allowed.

Message data - examples:

“robtarget;[[930,0,1455],[1,0,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]]”

“string;“Hello world!””

“num;23”

“bool;FALSE”

“bool{2, 2};[[TRUE,TRUE],[FALSE,FALSE]]”

“msgrec;[100,200]” (user defined data type)

The method `IpcMessage.SetData` is used to fill the `IpcMessage` with the appropriate data. Likewise, the `GetData` method retrieves the data from an `IpcMessage` object.

NOTE!

The `IpcMessage.Data` is set and retrieved as a byte array, `SetData(byte[] data)` and `byte[] GetData()`. This means you must convert the message data string to a byte array before calling the `SetData` method. It may look like this in C#:

```
Byte[] data = new UTF8Encoding().GetBytes("string;\"Hello world\"");
```

For more examples, see *Code example on page 122*.

NOTE!

The RAPID program can specify what RAPID data type it expects to receive by connecting it to a TRAP routine. A message containing data of a data type that no interrupt is connected to will be discarded with only an event log warning.

© Copyright 2010 ABB. All rights reserved.

RAPID Message Queue system parameters

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

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

Parameter	Description
RmqType	The following values are possible: • None - Disables the RAPID Message Queue functionality in this RAPID task. This is the default value. • Internal - Enables the RAPID Message Queue for local usage on the controller. • Remote - Enables the RAPID Message Queue for local usage and for PC and FlexPendant applications.
RmqMode	• Interrupt mode - A message can be received either by connecting a trap routine to a specified message type or by using the send-wait functionality. Any messages that are not the answer to an active send-wait instruction or have the type connected to a trap routine will be discarded. This is the default mode. • Synchronous mode - All messages will be queued and can only be received through the new read-wait instruction `RMQReadWait`. No messages will be discarded unless the queue is full. The send-wait instruction is not available in this mode. New mode from 5.12.
RmqMaxMsgSize	The maximum data size, in bytes, for a message. The default value is 350. The value cannot be changed in RobotStudio or on the FlexPendant.
RmqMaxNoOfMsg	Maximum number of messages in queue. The default value is 5. The value cannot be changed in RobotStudio or on the FlexPendant.



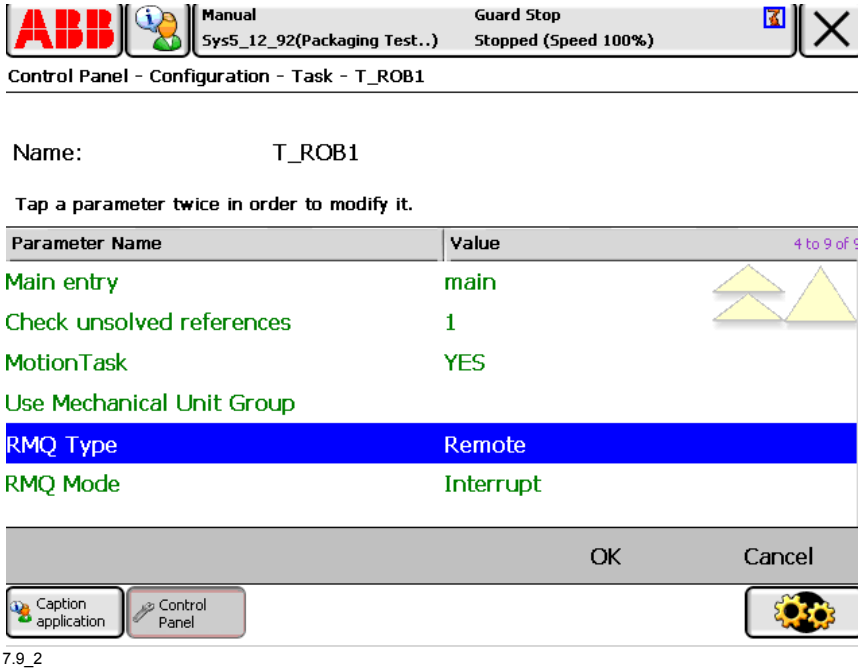
NOTE!

To read the values of these system parameter from the PC SDK you use the `IpcQueue` properties `RemoteAccessible, MessageSizeLimit` and `Capacity`.

© Copyright 2010 ABB. All rights reserved.

Remote RmqType

The system parameter *RmqType* must be set to **Remote** to enable messaging between RAPID and PC SDK:



7.9_2

© Copyright 2010 ABB. All rights reserved.

Code example

This simple messaging example can be tested with a virtual or a real controller. The system parameter *RmqType* must be set to **Remote** as shown in *RAPID Message Queue system parameters on page 120*.

The following code sample creates a message and sends it to a RAPID task, which reads it and sets a RAPID variable accordingly. Then an “Acknowledged” message is sent back to the PC SDK queue. Finally, the PC SDK application launches the received message in a Message Box.

PC SDK - C#

A message is created and sent to the RAPID queue “RMQ_T_ROB1”. An answer message is then received from RAPID and launched in a Message Box.

C#:

```
//declarations
private Controller c;
private IpcQueue tRob1Queue;
private IpcQueue myQueue;
private IpcMessage sendMessage;
private IpcMessage recMessage;
...

//initiation code, eg in constructor
c = new Controller(); //default ctrl used here (App.config)

//get T_ROB1 queue to send msgs to RAPID task
tRob1Queue = c.Ipc.GetQueue("RMQ_T_ROB1");

//create my own PC SDK queue to receive msgs
if (!c.Ipc.Exists("RAB_Q"))
{
  myQueue = c.Ipc.CreateQueue("PC_SDK_Q", 5, Ipc.IPC_MAXMSGSIZE);
  myQueue = c.Ipc.GetQueue("PC_SDK_Q");
```

© Copyright 2010 ABB. All rights reserved.

```
}

//Create IpcMessage objects for sending and receiving
sendMessage = new IpcMessage();
recMessage = new IpcMessage();
...

//in an event handler, eg. button_Click
SendMessage(true);
CheckReturnMsg();
...
public void SendMessage(bool boolMsg)
{
  Byte[] data = null;
  //Create message data
  if (boolMsg)
  {
     data = new UTF8Encoding().GetBytes("bool;TRUE");
  }
  else
  {
     data = new UTF8Encoding().GetBytes("bool;FALSE");
  }

  //Place data and sender information in message
  sendMessage.SetData(data);
  sendMessage.Sender = myQueue.QueueId;

  //Send message to the RAPID queue
  tRob1Queue.Send(sendMessage);
}
```

© Copyright 2010 ABB. All rights reserved.

```
private void CheckReturnMsg()
{
  IpcReturnType ret = IpcReturnType.Timeout;
  string answer = string.Empty;
  int timeout = 5000;

  //Check for msg in the PC SDK queue
  ret = myQueue.Receive(timeout, recMessage);

  if (ret == IpcReturnType.OK)
  {
     //convert msg data to string
     answer = new UTF8Encoding().GetString(recMessage.Data);
     MessageBox.Show(answer);

     //MessageBox should show: string;"Acknowledged"
  }
  else
  {
     MessageBox.Show("Timeout!");
  }
}
```

RAPID

A trap is created for a message of data type `bool`. In the trap, the value of the message data is assigned to the *flag* variable. Then an *"Acknowledged"* message is sent back to the PC SDK client. In *main* the WHILE loop is executed until a message with a TRUE value is received.

```
MODULE RAB_COMMUNICATION
   VAR bool flag := FALSE;
   VAR intnum connectnum;

   PROC main()
     CONNECT connectnum WITH RABMsgs;
     IRMQMessage flag, connectnum;
     WHILE flag = FALSE DO
        !do something, eg. normal processing...
        WaitTime 3;
     ENDWHILE
     !PC SDK message received - do something...
     TPWrite "Message from PC SDK, will now...";
     IDelete connectnum;
     EXIT;
```

© Copyright 2010 ABB. All rights reserved.

```
    ENDPROC

    TRAP RABMsgs
      VAR rmqmessage msg;
      VAR rmqheader header;
      VAR rmqslot rabclient;
      VAR num userdef;
      VAR string ack := "Acknowledged";

      RMQGetMessage msg;
      RMQGetMsgHeader msg \Header:=
           header\SenderId:=rabclient\UserDef:=userdef;

      !check data type and assign value to flag variable
      IF header.datatype = "bool" THEN
         RMQGetMsgData msg, flag;
         !return receipt to sender
         RMQSendMessage rabclient, ack;
      ELSE
         TPWrite "Unknown data received in RABMsgs...";
      ENDIF

    ENDTRAP
ENDMODULE
```

NOTE!

Error handling should be implemented in C# and in RAPID.

NOTE!

From RobotWare 5.12 there is a new RAPID instruction, `RMQEmptyQueue`, that can be used to empty the queue in a task.

© Copyright 2010 ABB. All rights reserved.

5.10. Messaging domain

© Copyright 2010 ABB. All rights reserved.

6 PC - Debugging and troubleshooting

6.1. Debugging

Introduction

Using the Visual Studio debugger for a PC SDK application presents no difference compared to standard .NET development. Debugging can be done using the virtual IRC5 in RobotStudio or a real controller.

Exception error codes

Some exceptions that may appear during development have error codes associated with them. The error codes may help you correct the problem.

Code	Description
0x80040401	The requested poll level could not be met, poll level low is used.
0x80040402	The requested poll level could not be met, poll level medium is used.
0xC0040401	No connection with controller.
0xC0040402	Error connecting to controller.
0xC0040403	No response from controller.
0xC0040404	Message queue full. (Should only happen if asynchronous calls are made.)
0xC0040405	Waiting for a resource.
0xC0040406	The message sent is too large to handle.
0xC0040408	A string does not contain characters exclusively from a supported encoding, for example ISO-8859-1 (ISO-Latin1).
0xC0040409	The resource can not be released since it is in use.
0xC0040410	The client is already logged on as a controller user.
0xC0040411	The controller was not present in NetScan.
0xC0040412	The NetScanID is no longer in use. Controller removed from list.
0xC0040413	The client id is not associated with a controller user. Returned only by methods that need to check this before sending request to controller. Otherwise, see 0xC004840F.
0xC0040414	The RobotWare version is later than the installed Robot Communication Runtime. A newer Robot Communication Runtime needs to be installed. Returned by RobHelperFactory.
0xC0040415	The major and minor part of the RobotWare version is known, but the revision is later and not fully compatible. A newer Robot Communication Runtime needs to be installed. Code returned by RobHelperFactory.
0xC0040416	The RobotWare version is no longer supported. Code returned by RobHelperFactory.
0xC0040417	The helper type is not supported by the RobotWare. Helper might be obsolete or for later RobotWare versions, or the helper may not be supported by a BootLevel controller. Code returned by RobHelperFactory.
0xC0040418	System id and network id mismatch, they do not identify the same controller.
0xC0040601	Call was made by other client than the one that made the Connect() call.
0xC0040602	File not found on the local file system. Can be that file, directory or device does not exist.

© Copyright 2010 ABB. All rights reserved.

Code	Description
0xC0040603	File not found on the remote file system. Can be that file, directory or device does not exist.
0xC0040604	Error when accessing/creating file on the local file system.
0xC0040605	Error when accessing/creating file on the remote file system.
0xC0040606	The path or filename is too long or otherwise bad for the VxWorks file system.
0xC0040607	The file transfer was interrupted. When transferring to remote system, the cause may be that the remote device is full.
0xC0040608	The local device is full.
0xC0040609	Client already has a connection and can not make a new connection until the present one is disconnected.
0xC0040701	One or more files in the release directory is corrupt and cannot be used when launching a VC.
0xC0040702	One or more files in the system directory is corrupt and cannot be used when launching a VC.
0xC0040703	A VC for this system has already been started; only one VC per system is allowed.
0xC0040704	Could not warm start VC since it must be cold started first.
0xC0040705	The requested operation failed since VC ownership is not held or could not be obtained.
0xC0048401	Out of memory.
0xC0048402	Not yet implemented.
0xC0048403	The service is not supported in this version of the controller.
0xC0048404	Operation not allowed on active system.
0xC0048405	The data requested does not exist.
0xC0048406	The directory does not contain all required data to complete the operation.
0xC0048407	Operation rejected by the controller safety access restriction mechanism.
0xC0048408	The resource is not held by caller.
0xC0048409	An argument specified by the client is not valid for this type of operation.
0xC004840A	Mismatch in controller id between backup and current system.
0xC004840B	Mismatch in key id, that is, options, languages and so on. between backup and current system.
0xC004840C	Mismatch in robot type between backup and current system.
0xC004840D	Client not allowed to log on as local user.
0xC004840F	The client is not logged on as a controller user.
0xC0048410	The requested resource is already held by caller
0xC0048411	The max number of the requested resources has been reached.
0xC0048412	No request active for the given user.
0xC0048413	Operation/request timed out on controller.
0xC0048414	No local user is logged on.
0xC0048415	The operation was not allowed for the given user.
0xC0048416	The URL used to initialize the helper does not resolve to a valid object.
0xC0048417	The amount of data is too large to fulfill the request.
0xC0048418	Controller is busy. Try again later.
0xC0048419	The request was denied.

© Copyright 2010 ABB. All rights reserved.

Code	Description
0xC004841A	Requested resource is held by someone else.
0xC004841B	Requested feature is disabled.
0xC004841C	The operation is not allowed in current operation mode. For example, a remote user may not be allowed to perform the operation in manual mode.
0xC004841D	The user does not have required mastership for the operation.
0xC004841E	Operation not allowed while backup in progress.
0xC004841F	Operation not allowed when tasks are in synchronized state.
0xC0048420	Operation not allowed when task is not active in task selection panel.
0xC0048421	Mismatch in controller id between backup and current system.
0xC0048422	Mismatch in controller id between backup and current.
0xC0048423	Invalid client id.
0xC0049000	RAPID symbol was not found.
0xC0049001	The given source position is illegal for the operation.
0xC0049002	The given file was not recognized as a program file, for example the XML semantics may be incorrect.
0xC0049003	Ambiguous module name.
0xC0049004	The RAPID program name is not set.
0xC0049005	Module is read protected.
0xC0049006	Module is write protected.
0xC0049007	Operation is illegal in current execution state.
0xC0049008	Operation is illegal in current task state.
0xC0049009	The robot is not on path and is unable to restart. Regain to or clear path.
0xC004900A	Operation is illegal at current execution level.
0xC004900B	Operation can not be performed without destroying the current execution context.
0xC004900C	The RAPID heap memory is full.
0xC004900D	Operation not allowed due to syntax error(s).
0xC004900E	Operation not allowed due to semantic error(s).
0xC004900F	Given routine is not a legal entry point. Possible reasons are: routine is a function, or routine has parameters.
0xC0049010	Illegal to move PCP to given place.
0xC0049011	Max number of rob targets exceeded.
0xC0049012	Object is not mod possible. Possible reasons are: object is a variable, object is a parameter, object is an array.
0xC0049013	Operation not allowed with displacement active.
0xC0049014	The robot is not on path and is unable to restart. Regain to path. Clear is not allowed.
0xC0049015	Previously planned path remains. Choose to either consume the path, which means the initial movement might be in an unexpected direction, or to clear the path and move directly to next target.
0xC004A000	General file handling error.
0xC004A001	The device is full.
0xC004A002	Wrong disk. Change disk and try again.
0xC004A003	The device is not ready.
0xC004A004	Invalid path.

© Copyright 2010 ABB. All rights reserved.

Code	Description
0xC004A005	Not a valid device.
0xC004A006	Unable to create directory.
0xC004A007	The directory does not exist.
0xC004A008	The directory already exists.
0xC004A009	The directory contains data.
0xC004A00B	Unable to create file.
0xC004A00C	File not found or could not be opened for reading.
0xC004A200	Disable of unit not allowed at trustlevel 0.

© Copyright 2010 ABB. All rights reserved.

6.2. Troubleshooting

Overview

If you encounter problems with your PC SDK application follow these steps before contacting ABB support.

	Action
1	Check whether your problem is in the *checklist* in the next section.
2	Answers to many questions are available in the Release Notes of the specific PC SDK release. The document is available on the RobotWare DVD and on the Software Download Site.
3	Pinpoint the problem by debugging your code so that a precise problem description can be provided.
4	Check the User Forum of ABB's RobotStudio Community, which includes a forum dedicated to discussion and chat on PC SDK topics.



TIP!

At *www.abb.com/roboticssoftware* there is a link to the *RobotStudio Community*.

Checklist

- Unable to connect to controllers? Make sure the system on the controller has the RobotWare option *PC Interface*. This applies to both virtual and real controllers.
- Is the problem is GUI hangings? Make sure you use `Invoke` when modifying the user interface due to a robot controller event. For more information, see *GUI and controller event threads in conflict on page 44* and *Invoke method on page 45*.
- Is the problem is related to netscan? If NetworkScanner.Scan does not find the robot controller during netscan you should try to increase the time allowed for scanning. Increase the *networks scanner delay* time in an app.config file as explained in *Application configuration file on page 35* or add the time directly in the code like this:
```
NetworkScanner aScanner = new NetworkScanner(); aScanner.Scan();
System.Threading.Thread.Sleep(4000);aScanner.Scan();
```
- Do you get "*Invalid Client ID*" when trying to do a read operation toward the robot controller? If so, the reason is probably that you have forgotten to log on to the controller. To *write* to RAPID data or to the configuration database, for example, you also need to require mastership. For more information, see *Logon and logoff on page 64* and *Mastership on page 33*.
- If you are working with a previous version of PC SDK (earlier than 5.10) you might run into problems related to licence verification? If you get the run-time error *"A valid license cannot be granted for the type ABB.Robotics.Controllers.Licenses.PCSdk. Contact the manufacturer of the component for more information"* when accessing the `NetworkScanner` and the `Controller` classes you need to add a licx file to the project. For more information, see *Licenses.licx on page 39*.

© Copyright 2010 ABB. All rights reserved.

Important support information

If you are unable to solve the problem, make sure that the following information is available when you contact ABB support:

- Written description of the problem.
- Application source code.
- System error logs.
- A backup of the system.
- Description of work-around if such exists.

© Copyright 2010 ABB. All rights reserved.

7 Deployment of a PC SDK application

7.1. Overview

Introduction

When your application is ready it has to be deployed to the customer's PC. This chapter gives information about the facilities for deployment included in the PC SDK installation.

NOTE!

Neither PC SDK nor a PC SDK licence need to be installed on the PC that will host your application. Furthermore, from PC SDK 5.10 you do NOT need to add the licence key to your project as described in *Licenses.licx on page 39*, as deployed PC applications no longer perform license verification when executing.

Facilities for deployment

In the *redistributable* folder at C:\Program Files\ABB Industrial IT\Robotics IT\SDK\PC SDK 5.xx there are some files to be used for deployment of a PC SDK application:

- ABBControllerAPI.msm
- ABB Industrial Robot Communication Runtime.msi

These two packages include all dependencies a PC SDK application has apart from .NET 2.0.

ABBControllerAPI.msm

A PC SDK application cannot execute without the PC SDK assemblies it references. For your convenience, the ABBControllerAPI merge module contains the PC SDK assemblies. Add it to your install program to have them installed in the Global Assembly Cache (GAC).

The GAC is automatically installed with the .NET runtime. It enables a PC to share assemblies across numerous applications. If the customer's PC has RobotStudio Online of the same release as the PS SDK used to create the application the PC SDK dlls your application needs should be in the GAC already.

NOTE!

If you want to create an msi file (or a setup.exe) of the msm file, you can include the ABBControllerAPI.msm file in a Visual Studio **SetUp Project**.

NOTE!

Before, ABBControllerAPI.msm worked only with *InstallShield*. This problem has now been resolved.

ABB Industrial Robot Communication Runtime.msi

To connect a PC SDK application to a controller either RobotStudio or Robot Communications Runtime is required. If RobotStudio is not installed on the PC that hosts your application, Robot Communications Runtime needs to be included in your installation. ABB Industrial Robot Communication Runtime.msi can be used for redistribution as a separate installation.

© Copyright 2010 ABB. All rights reserved.

A

Array 77
- dimensions 77
- enumerator 79
- indexers 77
- object 77

C

compatibility 40
Configuration 35
- App.config 35
- CAPI 35
- Section tag 35

ControllerInfo Object 62

E

Exception 48
- .Error Codes 127
- .NET 49
- Try-catch 48
- Typecasting 48

G

GAC 20

I

Installation 19
- Requirements 19

Invoke 45
IRC5 controller. 13

N

NetworkScanner 61
NetworkWatcher 62

R

ReadItem 80
Remote operator dialog 97

W

WriteItem 80

© Copyright 2010 ABB. All rights reserved.

© Copyright 2010 ABB. All rights reserved.