

Application manual  
FlexPendant SDK

5.13

Document ID: 3HAC036958-001

Revision: -

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

Overview .....	7
Product documentation, M2004 .....	9
Safety .....	11
<b>1 Introduction</b> .....	<b>13</b>
1.1 About creating controller applications .....	13
1.2 Documentation and help .....	15
1.3 Terminology .....	17
<b>2 Installation and development environment</b> .....	<b>19</b>
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 Visual Studio 2005 projects to Visual Studio 2008 .....	29
<b>3 Run-time environment</b> .....	<b>31</b>
3.1 Overview .....	31
<b>3.2 Running FlexPendant Applications.</b> .....	<b>32</b>
3.2.1 Components, assemblies and dlls .....	32
3.2.2 Deployment of FlexPendant application to a robot system .....	33
3.2.3 Communication between FlexPendant and controller .....	36
3.2.4 Understanding the FlexPendant application life cycle .....	37
3.2.5 FlexPendant TpsView attribute .....	39
3.2.6 ITpsViewSetUp and ITpsViewActivation .....	43
3.3 Release upgrades and compatibility .....	45
<b>4 Developing Controller applications</b> .....	<b>47</b>
4.1 Introduction .....	47
4.2 Analysis and design .....	48
4.3 Controller events and threads .....	50
4.4 User Authorization System .....	52
4.5 Exception handling .....	54
4.6 How to use the online help .....	57
<b>5 Using the FlexPendant SDK</b> .....	<b>59</b>
<b>5.1 Introduction</b> .....	<b>59</b>
5.1.1 About this chapter .....	59
5.1.2 System features supporting the use of customized screens .....	60
<b>5.2 Setting up a new project.</b> .....	<b>62</b>
5.2.1 Using the project template in Visual Studio .....	62
5.2.2 Setting up design support for FlexPendant controls .....	65
<b>5.3 Building the user interface.</b> .....	<b>66</b>
5.3.1 Introduction to visual design support .....	66
5.3.2 GUI controls and memory management .....	76
5.3.3 Container style .....	81
5.3.4 Command bar .....	85
5.3.5 FlexPendant fonts .....	87
5.3.6 The use of icons .....	88
5.3.7 TabControl .....	90
5.3.8 Button, TextBox and ComboBox .....	93
5.3.9 AlphaPad .....	94

# Table of Contents

---

5.3.10 ListView . . . . .	97
5.3.11 CompactAlphaPad and NumPad . . . . .	99
5.3.12 GTPUMessageBox . . . . .	100
5.3.13 GTPUFileDialog . . . . .	102
5.3.14 Data of RAPID data and IO signals . . . . .	105
<b>5.4 Launching other views . . . . .</b>	<b>111</b>
5.4.1 Using launch service . . . . .	111
5.4.2 Using standard dialog box to modify data . . . . .	114
<b>5.5 Using the Controller API . . . . .</b>	<b>116</b>
5.5.1 ABB.Robotics.Controllers . . . . .	116
5.5.2 Accessing the controller . . . . .	119
<b>5.5.3 Rapid domain . . . . .</b>	<b>122</b>
5.5.3.1 Working with RAPID data . . . . .	122
5.5.3.2 Handling RAPID arrays . . . . .	130
5.5.3.3 ReadItem and WriteItem methods . . . . .	133
5.5.3.4 UserDefined data . . . . .	134
5.5.3.5 RAPID symbol search . . . . .	139
5.5.3.6 RAPID execution . . . . .	145
5.5.3.7 Modifying modules and programs . . . . .	147
5.5.4 IO system domain . . . . .	150
5.5.5 Event log domain . . . . .	156
5.5.6 Motion domain . . . . .	158
5.5.7 File system domain . . . . .	161
5.5.8 System info domain . . . . .	163
<b>6 Robust FlexPendant applications . . . . .</b>	<b>165</b>
6.1 Introduction . . . . .	165
6.2 Memory management . . . . .	167
6.3 Performance . . . . .	170
6.4 Reliability . . . . .	174
<b>7 FlexPendant - Debugging and troubleshooting . . . . .</b>	<b>179</b>
7.1 Debugging . . . . .	179
7.2 Debug output . . . . .	183
7.3 Debugging the virtual FlexPendant . . . . .	186
7.4 Debugging the FlexPendant device . . . . .	190
7.5 Troubleshooting FlexPendant applications . . . . .	193
<b>8 Localizing a FlexPendant application . . . . .</b>	<b>197</b>
8.1 Adding support for several languages . . . . .	197
<b>9 Deployment of a FlexPendant SDK application . . . . .</b>	<b>207</b>
9.1 Overview . . . . .	207
9.2 Deployment of an application without a license . . . . .	208
9.3 Deployment of a licensed application . . . . .	211
9.4 Deployment using FTP . . . . .	214
<b>Index . . . . .</b>	<b>215</b>

---





---

## Overview

---

### About this manual

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

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

---

### Usage

FlexPendant SDK application manual covers application development using the FlexPendant SDK. FlexPendant SDK is for developing end user applications that are FlexPendant based. PC SDK is a tool for developing end user applications that are PC based (PC implies a Windows machine in this context). For more information on PC SDK, see *Application manual PC SDK*. The process of actually developing the application involves using a PC with Visual Studio for both PC SDK and FlexPendant SDK applications.

---

### Who should read this manual?

This manual is mainly intended for developers, who use FlexPendant SDK to create robot applications adapted to end-user needs. 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 are preferred languages. Visual J# and Visual C++ can also be used.
- be used to object oriented programming.

*Continues on next page*

## 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 FlexPendant applications. Life cycle of a FlexPendant application. Upgrades and compatibility.
4.	Developing FlexPendant SDK applications. Analysis and design. Important programming issues: controller events and threads, UAS, exception handling. Online help.
5.	Using the FlexPendant SDK. Visual design support. GUI controls. Launching standard views. . How to add controller functionality using the Controller API. Programming issues and code samples in VB and C#.
6.	How to develop well performing and robust FlexPendant applications. Memory management, performance and reliability. Exception handling.
7.	Testing, debugging and troubleshooting FlexPendant SDK applications. Using printouts, error codes in exceptions and so on. Checklist for contacting a service organization.
8.	How to add support for several languages to a custom FlexPendant application.
9.	How to create an additional option and how to make a product of a FlexPendant application.

## References

Reference	Document ID
Operating Manual - IRC5 with FlexPendant	3HAC16590-1
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
Application Manual - PC SDK	3HAC036957-001

## Revisions

Revision	Description
-	First edition From FlexPendant SDK 5.13 onwards this manual replaces: <i>Application Manual - Robot Application Builder (3HAC028083-001)</i> For information on PC SDK refer, <i>Application manual - PC SDK</i>



---

## Product documentation, M2004

---

### General

The robot documentation can 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 can be ordered from ABB. The documents listed are valid for M2004 robot systems.

However, all documents listed are 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

*Continues on next page*

*Continued*

---

## **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**

## 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*.



# 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.

FlexPendant 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 added to the standard views of the FlexPendant.

For PC based applications, see *Application manual - PC SDK*.



#### NOTE!

Controller applications are not platform independent. You must choose to develop the application for either the FlexPendant or the PC platform (For more information, see *Application manual - PC SDK*).

### Local vs Remote client

The difference between the two platforms is that a FlexPendant application is a local client, whereas a PC application is a remote 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 FlexPendant and controller on page 36*.

### 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.

FlexPendant SDK enables customized user interfaces for IRC5. It is important to keep in mind, however, that FlexPendant SDK itself does not guarantee increased customer value. To achieve this, FlexPendant 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*

# 1 Introduction

---

## 1.1. About creating controller applications

*Continued*

---

### **.NET and Visual Studio**

FlexPendant SDK uses Microsoft .NET and Microsoft Visual Studio. It is thus assumed that you know how to program for .NET 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. Most prefer C# or Visual Basic, which both offer safe and efficient development.

For a Windows programmer familiar with Visual Studio and .NET, developing a customized operator view is rather straight-forward. FlexPendant SDK is fully integrated with Visual Studio, which means that a .NET programmer will recognize wizards for project setup and tools for visual design support and debug and so on.

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!**

Knowledge of Windows programming, object orientation and a .NET programming are the prerequisites to use FlexPendant SDK.

---

### **Robustness and performance**

Developing an application for the FlexPendant, a device with limited process and memory resources, can be quite demanding. Issues such as performance and memory management need to be addressed.

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



#### **NOTE!**

Take the time to study this manual along with the release notes, and avoiding rushing into coding.

---

## 1.2. Documentation and help

---

### Introduction

FlexPendant SDK includes an extensive on-line help module, which comes with the installation of the product. 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 FlexPendant SDK*
- *Reference Manual FlexPendant SDK*
- *FlexPendant StyleGuide*

---

### Application manual

This *Application manual - FlexPendant SDK*, is the recommended way to get started if you are new to FlexPendant SDK development. It explains how FlexPendant SDK works. It has code examples in C# and VB 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 manual PDF can be found in the installation directory, at C:\Program Files\ABB Industrial IT\Robotics IT\SDK\FlexPendant SDK 5.xx.

---

### SDK Reference Help

The *Reference Manual FlexPendant SDK* should be used while programming. It makes up the complete reference to the FlexPendant SDK class libraries. 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 FlexPendant 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 FlexPendant SDK related issues.

---

### FlexPendant StyleGuide

Good usability is achieved when the program itself communicates possible actions and how to perform them. To encourage careful design of the visual appearance the *FlexPendant StyleGuide* is also part of the installation. It is ABB Robotics' best practices for visual design of the FlexPendant user interface.

*Continues on next page*

# 1 Introduction

---

## 1.2. Documentation and help

*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 FlexPendant 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](http://www.abb.com/robotics) > *RobotStudio Community*.

---

### MSDN

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



## 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 in the following table.

### Definitions

Term	Definition
.NET Compact Framework (.NET CF)	Version of Microsoft's .NET framework providing the run-time environment for applications running on embedded devices, such as the FlexPendant. It includes a class library, which is almost a subset of the rich .NET framework for the desktop.
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 the FlexPendant 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.
FlexPendant SDK programmer	A programmer who uses FlexPendant SDK to develop custom applications.
FlexPendant SDK application	A custom application developed with FlexPendant SDK.
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.

*Continues on next page*

# 1 Introduction

## 1.3. Terminology

Continued

Term	Definition
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.
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
FlexPendant SDK	FlexPendant Software Development Kit
GUI	Graphical User Interface
MSDN	Microsoft Developer Network, source of information for .NET developers at: <a href="http://www.msdn.com">www.msdn.com</a>
SDK	Software Development Kit
TAF	Teach Pendant Application Framework, all applications using the FlexPendant SDK must run as TAF clients. For more information, see <a href="#">TAF - Application host framework on page 37</a> .
TCP/IP	Transmission Control Protocol (TCP) and Internet Protocol (IP)
VB	Visual Basic
VS	Visual Studio

© Copyright 2010 ABB. All rights reserved.

## 2 Installation and development environment

### 2.1. Installation overview

---

#### About this section

This section describes how to install FlexPendant SDK. When the installation is complete, you can program, compile and test FlexPendant 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: VS 2005 (Standard Edition or better) or VS 2008 (Professional Edition or better ).
- .NET Compact Framework 2.0 Service Pack 1 or 2

The following hardware requirement have to be met:

- 50 MB free disk-space on the installation disk

Both FlexPendant generations are supported:

- SxTPU-1, which executes with .NET CF 2.0 and WinCE 4.2.
- SxTPU-2, which executes with .NET CF 2.0 and WinCE 5.0.
- SxTPU3, which executes with NET CF 3.5 and Windows CE 6.0.



#### **NOTE!**

The controller system must have the RobotWare option FlexPendant Interface



#### **NOTE!**

FlexPendant 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.

## 2 Installation and development environment

### 2.1. Installation overview

*Continued*

#### Requirements for installing and using FlexPendant SDK

FlexPendant SDK is installed while you install RobotStudio. For more information on installing RobotStudio, see *Installing and Licensing RobotStudio* in *Operating manual - RobotStudio*. To use FlexPendant 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...
installing RobotStudio	install Microsoft Visual Studio 2005 or 2008.
debugging using a virtual IRC5	learn how to run the virtual IRC5 in RobotStudio.
debugging using the real FlexPendant device	install the <i>.NET Compact Framework 2.0 Service Pack 1 or 2</i> , which can be downloaded from <a href="http://www.microsoft.com">http://www.microsoft.com</a> . The <i>User Forum of RobotStudio Community</i> has information on how to attach the Visual Studio debugger to the device. For more information, see also <a href="#">Debugging the FlexPendant device on page 190</a> .
executing the application targeting a real IRC5 system	check that the robot system has the controller option <i>FlexPendant Interface</i> (for FlexPendant applications). Set up a connection between your PC and the robot controller. For more information, see <a href="#">How to set up your PC to communicate with robot on page 23</a> for details about how this is done.



#### NOTE!

The Visual Studio installation installs .NET and Compact Framework 2.0.

#### About the FlexPendant SDK installation

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

##### RAB 5.11 to 5.12

RAB 5.11 and later installs PC SDK and FlexPendant SDK side by side with any previously installed versions.

##### 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 > FlexPendant SDK Overview.

© Copyright 2010 ABB. All rights reserved.

*Continues on next page*

---

#### What is installed?

The installation generates the following features on your PC:

- *SDK assemblies and resources*
- *Application manual FlexPendant SDK*
- *FlexPendant Style Guide*
- *Reference Manual FlexPendant SDK*

---

#### Working with several versions

A FlexPendant SDK application normally targets a specific RobotWare release. Assuming you are developing a FlexPendant SDK application for a new customer who uses RobotWare 5.12 and simultaneously you are maintaining an existing FlexPendant SDK application for a customer with robot system using RobotWare 5.09, then you will need to work with two different FlexPendant SDK releases on your PC. For more information about releases and compatibility, see [Release upgrades and compatibility on page 45](#).

#### FlexPendant applications

If you install FlexPendant SDK 5.13 and previous versions of FlexPendant SDK which came along with PC SDK as Robot Application Builder, the previous versions will exist on your PC. You choose which FlexPendant SDK version to use when you set up your application project in Visual Studio. For more information, see [Using the project template in Visual Studio on page 62](#).

You should make sure that the FlexPendant SDK GUI controls the Visual Studio Designer uses is of the same version. If you have worked with another FlexPendant SDK version before, you will need to remove the GUI controls that you have added to the Visual Studio Toolbox and then add them again, pointing to the correct version in the browser. For more information, see [Setting up design support for FlexPendant controls on page 65](#).

---

#### Installation procedure

The installation procedure is very simple. By default FlexPendant 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 FlexPendant SDK, or remove FlexPendant SDK from RobotStudio select Custom in the RobotStudio installation wizard and select or unselect the feature FlexPendant SDK.

#### **NOTE!**

You are also strongly advised to read 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.



## 2 Installation and development environment

---

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

#### 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 allowed 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 <b>License Install Wizard</b> .
3	Follow the instructions in the wizard.



##### NOTE!

To execute FlexPendant SDK 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](#).

## 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.

---

### 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.

The connection is also used for downloading a FlexPendant application to the controller file system and test it on the real FlexPendant device.

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



## 2 Installation and development environment

---

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

*Continued*

---

#### Serial connection to the Console port

In order to use the console window of the IRC5 controller you can connect your PC with a serial null-modem cable to the console port of the controller.



#### **NOTE!**

The cable must be a twisted cable with a female DB-9 connector in each end..

---

#### 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	<i>Windows Help - Configure TCP/IP settings.</i>
How to connect the PC to the Controller service port	<i>Connecting a PC to the Service Port in the RobotStudio help.</i>



## 2.4. Development environment

### Overview

This section presents an overview of the development environment used to create FlexPendant SDK applications. You can program and debug the application using Microsoft Visual Studio 2005 or 2008.

### 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!

For information about upgrading an existing FlexPendant SDK project to Visual Studio 2008 project, see *Conversion of Visual Studio 2005 projects to Visual Studio 2008 on page 29*.

### Visual design support and Databinding

The .NET Compact Framework 2.0. can be used for building enhanced user interfaces. FlexPendant specific controls are available in the Visual Studio toolbox since FlexPendant SDK 5.08 version.

Databinding is the process of binding a property of a GUI control to a data source, so that the property automatically reflects the value of the data source.

### 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 FlexPendant SDK applications only C# and Visual Basic are supported. Likewise, in this manual there are code samples in C# and Visual Basic, 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.

*Continues on next page*

## 2 Installation and development environment

---

### 2.4. Development environment

*Continued*

---

#### Integration with Visual Studio

When FlexPendant SDK is installed on your computer, it is integrated with Visual Studio. You will notice when starting a new project, for example, that the project type *FlexPendant* is available in the **New Project** window. When using the wizard to create a FlexPendant project, common SDK references are added to the project and some code is automatically generated.

The visual design support for the FlexPendant will be accessible from the **Toolbox** in Visual Studio and work the same way as the design support for an ordinary Windows application. As you will see, using FlexPendant 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 FlexPendant SDK related this will not help you.



#### **TIP!**

The *Reference Manual FlexPendant SDK* is found by clicking **Start** menu, then pointing at Programs > ABB Industrial IT > Robotics IT > RobotStudio 5.xx > SDK > Reference Manual FlexPendant SDK. Keep the reference file open while programming, as you will be needing it all the time.

## 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)
- Microsoft Visual Studio 2005 or 2008
- Controller option FlexPendant Interface

Controller 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 FlexPendant SDK selected)
- Microsoft Visual Studio 2005 or 2008
- Controller option PC Interface and FlexPendant 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

Using the virtual environment a FlexPendant application executes on the Virtual FlexPendant as an assembly (dll). You start the application from the ABB menu of the Virtual FlexPendant like you start it on the real FlexPendant.

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. For more information, see [Debugging the virtual FlexPendant on page 186](#).

*Continues on next page*

## 2 Installation and development environment

---

### 2.5. Two development models - virtual and real

*Continued*

---

#### Real tests necessary

The virtual environment is a very convenient choice, especially for testing and debugging. You should be aware however, that the virtual FlexPendant is more forgiving than the real device. Using only the virtual FlexPendant, it is very easy to neglect the restraints on memory consumption imposed by the real device. Images, for example, can easily consume all the FlexPendant memory available for custom applications.

This means that potential problems may be hard to detect until you test the application using a real robot system. It is almost as easy to debug code running on the real FlexPendant device. For more information, see [Debugging the FlexPendant device on page 190](#).

You should also be aware that your application shares CPU, memory and application host with all other FlexPendant applications. This means that a custom application can impact the overall performance of the FlexPendant.



#### NOTE!

Before shipping a FlexPendant application, it has to be tested properly, using a real system. Relying only on the virtual environment is far too risky. For more information, see [Robust FlexPendant applications on page 165](#).

---

#### Porting the application from virtual to real IRC5

A FlexPendant application that runs perfectly on the Virtual FlexPendant, but not on the real device since there can be a lag in response time due to TCP/IP communication, but the main problem is limited resources on the device, both memory and processor power.

The FlexPendant SDK does not slow down performance. Therefore your application is supposed to perform like any standard application of the FlexPendant. For more information on how to speed up a slow FlexPendant application, see [Performance on page 170](#).

---

#### 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 FlexPendant SDK application on page 207](#).

## 2.6. Conversion of Visual Studio 2005 projects to Visual Studio 2008

---

### Overview

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



#### **NOTE!**

For FlexPendant SDK projects one needs to manually edit the post-build event that builds the \*gtpu.dll. Find *Build Events* in the *Project Properties* and adapt the path to *vcvarsall.bat* to the new development environment.

## 2 Installation and development environment

---

### 2.6. Conversion of Visual Studio 2005 projects to Visual Studio 2008

## 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 FlexPendant SDK.

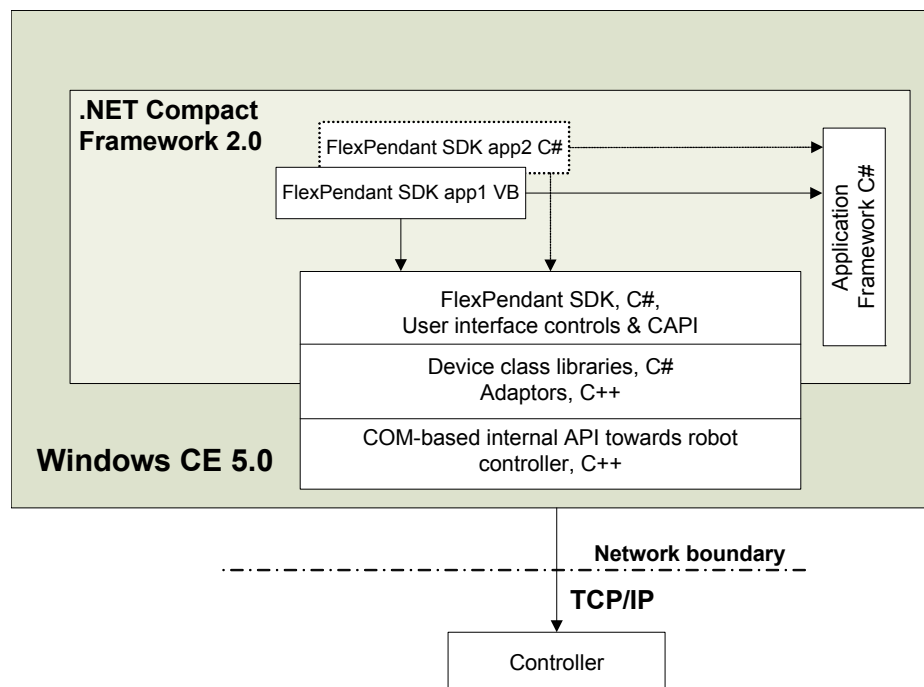
Application configuration is detailed as well as deployment of a FlexPendant application to a robot controller. The life cycle of a FlexPendant application is also explained.

#### Software architecture

The FlexPendant is an integral part of IRC5, yet a complete computer in itself. It has been developed with Microsoft's software technology for embedded systems, Windows CE and .NET Compact Framework, which is a subset of the full .NET Framework that the PC uses.

#### FlexPendant platform

The following illustration shows the Software architecture of FlexPendant SDK. Two FlexPendant applications, one using VB and the other C#, developed on top of the FlexPendant SDK. The FlexPendant SDK CAPI is the public API offering controller functionality. All communication with the robot controller is done via the internal Robot Communication Runtime. The FlexPendant SDK provides ABB made UI controls suitable for the FlexPendant.



3.1\_2

#### Controller API

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

## 3 Run-time environment

---

### 3.2.1. Components, assemblies and dlls

## 3.2 Running FlexPendant Applications

### 3.2.1. Components, assemblies and dlls

---

#### Building blocks

The .NET Framework is a library of components and supporting classes and structures, designed to make component development easy and robust. Components are packaged in assemblies, also called dlls.

Assemblies are the building blocks of .NET applications. An assembly is a reusable collection of types and resources, which are built to work together and form a logical unit of functionality. The simplest assembly is a single executable.

---

#### One or several assemblies

A FlexPendant project compiles to a dll, which cannot run as an independent application, but needs to be started by the Teach Pendant Application Framework (TAF), the application manager of the FlexPendant. For more information, see [Understanding the FlexPendant application life cycle on page 37](#).

In a normal case, a custom application for the FlexPendant is developed as a single component, but it is also possible to separate functionality into several components. This way the application will consist of several dlls. The reason for doing so might be one of the following:

- The amount of code is substantial. A modular design with small and well tested building blocks put together is one way of handling complexity.
- Different developers are working on the same custom application. For reasons of efficiency, they can split the functionality between them and work on one component each.



#### NOTE!

You can use different programming languages for different components.



### 3.2.2. Deployment of FlexPendant application to a robot system

#### Proxy assembly

When you compile a FlexPendant SDK application an additional assembly named \*gtpu.dll is automatically created. This is done by a tool, the *ABB Compliance Tool*, which verifies that the application complies to the FlexPendant requirements. This proxy dll is necessary to run the application on the FlexPendant.

To test the application on a real FlexPendant both assemblies must be downloaded to the SYSTEM or HOME directory of the robot controller. After this the FlexPendant should be restarted. At startup it loads the application assemblies from the controller.



#### TIP!

An advantage of deploying the dlls to the HOME directory is that they will be included in a system backup.

#### Download to real controller

To download your application assemblies to the controller you can use the *Online* function *File Transfer* of RobotStudio.

Another way of downloading the application to the robot controller is using the *ABB Compliance Tool*.

Step	Action
1	Verify that the following requirements are met before starting the procedure: <ul style="list-style-type: none"> <li>A network connection between the PC and the controller has to be configured, either using the controller LAN port or the controller service port. For further information see <a href="#">How to set up your PC to communicate with robot on page 23</a>.</li> <li>The RobotWare option <i>FlexPendant Interface</i> is required to run the application on a real system. Without the option you will not see the application in the ABB menu.</li> </ul>
2	Open Windows Explorer.
3	Start abbc.exe at C:\Program Files\ABB Industrial IT\Robotics IT\SDK\Flex-Pendant SDK 5.xx..
4	Click <b>Browse</b> and locate your Visual Studio project and the application assembly in the \bin\debug (or \bin\release) sub-folder.

## 3 Run-time environment

### 3.2.2. Deployment of FlexPendant application to a robot system

Continued

Step	Action
5	Check the <b>Deploy</b> box, enter the <b>IP Address</b> of the robot controller and click <b>Execute</b> . Deployment is done to the current system of the controller.  <b>Note!</b> If the application consists of several assemblies you need to repeat the procedure for all of these.
6	Restart the FlexPendant. For more information about different ways to do this, see <a href="#">Restart the FlexPendant on page 35</a> .

#### Using the command window

The *ABB Compliance Tool* can be used via the command window instead of the graphical interface. To do so you write:

```
abbct.exe /deploy="192.168.8.192" <PATH>\TpsViewHelloWorld.dll  
          <PATH>\TpsViewHelloWorld.gtpu.dll.
```

It is also possible to perform build and deployment in one step. To do this the deploy argument should come last:

```
abbct.exe <PATH>\TpsViewHelloWorld.dll /deploy="192.168.8.192"
```

Both the application and proxy assembly are deployed to the controller after the build.

#### FTP deployment

You can also use an FTP client to transfer files from your PC to the robot controller.

To use FTP you need:

- FTP client program
- Configured connection to the controller
- RobotWare option *FTP and NFS Client*
- Name and password

Step	Action
1	Transfer resources (for example icons) and the application and proxy assemblies to the <i>HOME</i> or <i>SYSTEM</i> directory of the current system of the controller.
2	Restart the FlexPendant. See the next section for information on how to do so.



#### NOTE!

For deployment to a customer when the application is ready, see [Deployment of a FlexPendant SDK application on page 207](#).

---

#### Restart the FlexPendant

If you want to restart the FlexPendant device without restarting the controller, choose one of these alternatives:

Alternative	Action
1	Write the command <code>fpcmd "-restart"</code> in the controller console window on your PC.
2	Perform the following sequence while holding the FlexPendant joystick: Move the joystick three times to the right, once to the left and once down. Confirm your wish to reset the FlexPendant in the dialog that will appear.
3	Unplug and plug the FlexPendant (power on/ power off). <b>Note!</b> This activates emergency stop.

---

#### Deploy application to virtual IRC5

Follow these steps to deploy an application to the Virtual FlexPendant:

Step	Action
1	Copy application and proxy assemblies and images to the HOME directory of the system you want to use on your PC. <b>Note!</b> Close the virtual FlexPendant first if you are <i>replacing</i> assemblies.
2	Restart the virtual FlexPendant.



#### TIP!

If you have problems running your application, try to put all files (dlls, gif files and so on.) in the vcbn directory of the robotware your system uses. This setup is as close to the real system setup as you can get.

## 3 Run-time environment

---

### 3.2.3. Communication between FlexPendant and controller

### 3.2.3. Communication between FlexPendant and controller

---

#### COM technology

The FlexPendant SDK uses an internal Controller API based on COM technology to communicate with the controller. This API uses sockets and TCP/IP (for more information, see [About terms and acronyms on page 17](#)) towards both real and virtual controllers.

Calls from the FlexPendant SDK to the controller are synchronous, that is, are done immediately through the COM servers. This increases execution speed and causes less overhead, which is important on a resource limited device.

---

#### 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 FlexPendant SDK cannot meet hard real-time demands. This is for several reasons:

- Part of the API executes on a non-real-time operating system.
- The controller sometimes has tasks to perform that have a higher right of priority.



#### NOTE!

The FlexPendant SDK does not affect performance in a negative way. You should expect your custom application to perform like any other application on the ABB menu.

#### 3.2.4. Understanding the FlexPendant application life cycle

---

##### Overview

Understanding the FlexPendant application life cycle improves your ability to design and debug the application.

##### TAF - Application host framework

The Teach Pendant Application Framework (TAF) is the application service provider that runs on the FlexPendant. It targets .NET CF and is implemented in C#. All applications using the FlexPendant SDK must run as TAF clients, as TAF contains services for hosting controls and for managing applications.

TAF uses a customized configuration file to create the appearance and behavior of the hosted application. It also defines a set of rules that have to be followed.

##### Starting a custom application

When the FlexPendant starts up TAF is already in the flash memory. Applications that will execute in the TAF container are now loaded from the controller.

If the FlexPendant SDK application is to be started manually by the end-user the application icon and text are placed in the ABB Menu. The other alternative is to have it started automatically by TAF at FlexPendant startup. For more information on how this is configured, see *FlexPendant TpsView attribute on page 39*.

##### Application life cycle

TAF handles the life cycle of a custom application, starting by calling the constructor of its TpsView class. After this, the `Install` method and then the `Activate` method in the same class execute.

During its lifetime, the application switches between the active and the passive state. Each time, either `Activate` or `Deactivate` is executed. In its active state the application is visible in the client view, in the passive state another application may have been opened or you may have opened another application through the FlexPendant task bar.

When the application is closed via the close button, first the `Deactivate` method runs and then `Uninstall`. After this the `Dispose` method of the TpsView class is called. Then the application instance is disposed of by TAF. For more information about how you can implement these methods, see *ITpsViewSetup and ITpsViewActivation on page 43*.

### 3 Run-time environment

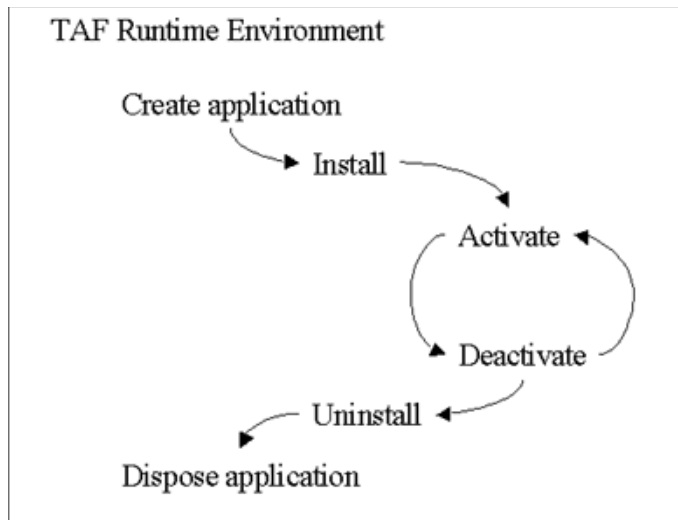
---

#### 3.2.4. Understanding the FlexPendant application life cycle

Continued

---

#### Illustration



4.3.3\_1

The figure illustrates the life cycle of a FlexPendant application.

---

#### Limited resources

As the FlexPendant is a device with very limited resources compared to a PC, you should learn and use the mechanisms implemented to assist you in writing efficient code.

Both process power and memory resources are limited compared to the virtual environment on the desktop. An application that runs very well on the virtual FlexPendant, may encounter serious problems in the real environment because of these limitations. For more information, see [Technical overview of the FlexPendant device on page 165](#).



#### NOTE!

You are strongly recommended to read [Robust FlexPendant applications on page 165](#) in this manual before starting coding. Always keep the limitations of the device in mind when you develop custom FlexPendant applications.

### 3.2.5. FlexPendant TpsView attribute

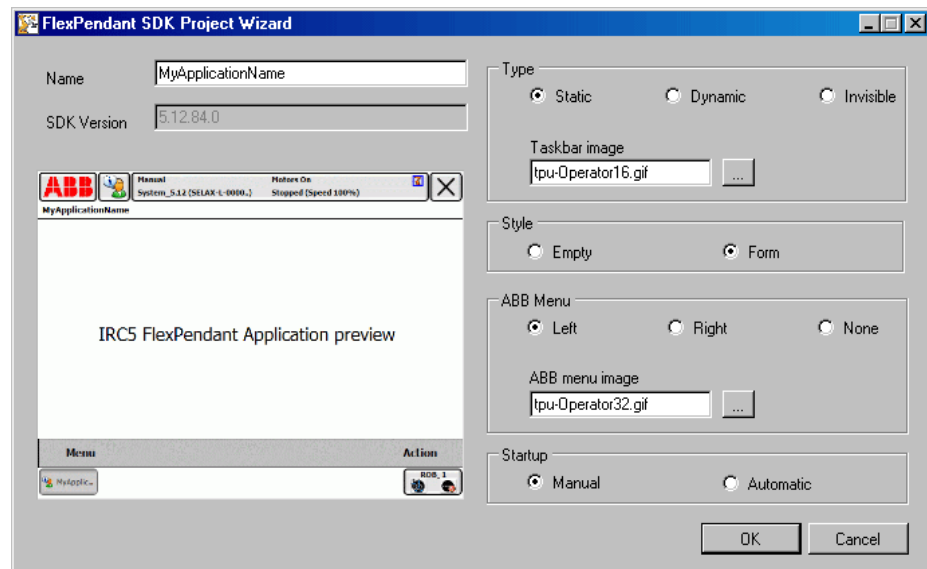
#### Overview

A custom FlexPendant application must implement the assembly attribute `TpsView`. For example, it is used to determine the visual appearance of the application in the ABB menu. The `TpsView` attribute is auto generated and located before the namespace definition in the application `view` class, that is, the class that displays the first view of a FlexPendant SDK application.

In this section all parameters of the `TpsView` attribute will be detailed.

#### Project wizard settings

The `TpsView` attribute is auto generated according to your settings in the **FlexPendant SDK Project Wizard** in Visual Studio:



6.2.2\_1

In C# the settings of the figure will render this code:

```
[assembly: TpsView("MyApplicationName", "tpu-Operator32.gif",
    "tpu-Operator16.gif", "TpsViewHelloWorld.dll",
    "TpsViewHelloWorld.TpsViewHelloWorld",
    StartPanelLocation.Left, TpsViewType.Static, StartupType =
    TpsViewStartupTypes.Manual)]
```

#### NOTE!

You can edit your settings directly in the auto generated code.



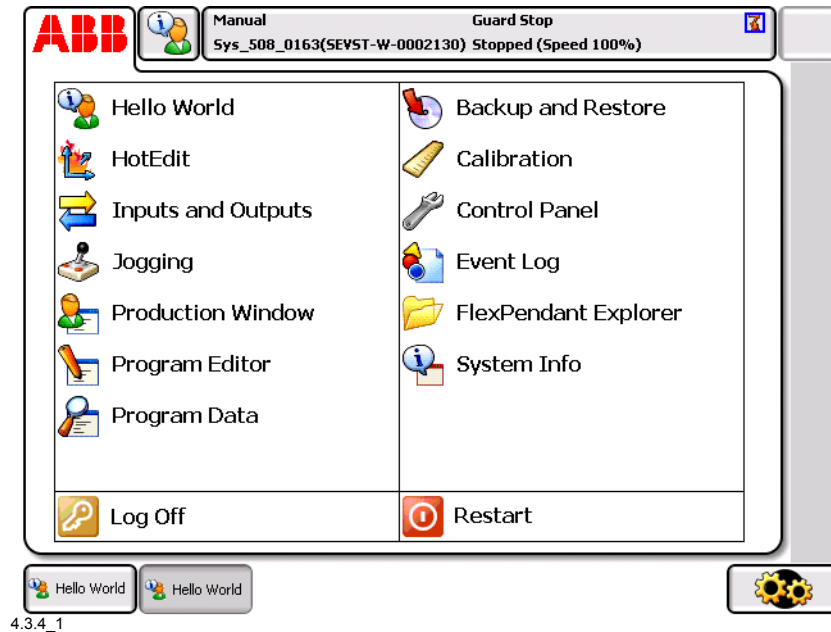
## 3 Run-time environment

### 3.2.5. FlexPendant TpsView attribute

Continued

#### Visual appearance

In run-time the most obvious result of this code is the way the custom application is presented in the ABB menu. In the following example, the first `TpsView` parameter has been changed from “MyApplicationName” to “Hello World”.



#### Application name

The first parameter of `TpsView` is the application name as it should appear in the ABB menu and on the task bar. The example uses “HelloWorld” as the application name.

#### Application icon

The second parameter is the file to be used as the application icon in the ABB menu. Usually a customized icon is used for the application. The example uses the default icon: “`tpu-Operator32.gif`”.

#### TaskBar icon

The third parameter is the file to be used as the application task bar icon. The example uses the default icon “`tpu-Operator16.gif`”.

#### Application assembly

The fourth parameter is the assembly name. If you change the name of the application assembly you also need to change this parameter to reflect that change. In the example, the assembly name is “`TpsViewHelloWorld.dll`”.

#### NOTE!

The assembly name must start with “`TpsView`” for TAF to identify it as an application to be loaded. If you forget this an error will be generated by the ABB verification tool when you try to build the project.





---

**Class name**

The fifth parameter specifies the fully qualified class name of the initial view of your application, which you chose in the **New Project** dialog box. In the example, the fully qualified class name is "TpsViewHelloWorld.TpsViewHelloWorld".

---

**Application location**

The sixth parameter determines the location of the application icon and text in the ABB menu. In the example these are displayed in the left column: `startPanelLocation.Left`.

Parameter value	Result
<code>StartPanelLocation.Left</code>	application visible to the left in the ABB menu.
<code>StartPanelLocation.Right</code>	application visible to the right in the ABB menu.
<code>StartPanelLocation.None</code>	application is not visible at all in the ABB menu.

**NOTE!**

`StartPanelLocation.None` was introduced in 5.11.01. Applications that use it can therefore NOT be run on RobotWare releases older than 5.11.01.

---

**Application type**

As you can tell from the two Hello World icons visible in the task bar, two instances of the Hello World application have been started. To enable this the seventh parameter is changed to `: TpsViewType.Dynamic`. Possible view type values are shown in the table:

Parameter value	Result
<code>TpsViewType.Dynamic</code>	You can start multiple instances of the application.
<code>TpsViewType.Static</code>	You can only start one instance of the application.
<code>TpsViewType.Invisible</code>	A background application with no GUI and no icon on the task bar. Can be started automatically or manually. Only one instance is possible.

**NOTE!**

Unless there is special need for it, you should allow the user to start only one instance of the application: `TpsViewType.Static`. The reason is that working with several instances takes up valuable memory resources.

---

**Startup type**

The eighth parameter determines how the application is started. In the example the startup type is `TpsViewStartupTypes.Manual` and the application is started from the ABB menu. Using the manual startup type it is also possible to have the application started by RAPID or at operating mode change to auto for example. For more information, see [System features supporting the use of customized screens on page 60](#)

If `TpsViewStartupTypes.Automatic` is chosen, the application is started automatically by TAF whenever the FlexPendant is restarted or a new user logs on.

## 3 Run-time environment

---

### 3.2.5. FlexPendant TpsView attribute

*Continued*

---

#### Related information

For information about the **Style** setting of the **Project Wizard**, see [Container style on page 81](#).

To find the **FlexPendant SDK Project Wizard** in Visual Studio, see [Setting up a new project on page 62](#).

#### 3.2.6. ITpsViewSetUp and ITpsViewActivation

---

##### ITpsViewSetUp

An application that TAF needs to initialize must have a default (empty) constructor and must implement the interface `ITpsViewSetUp`, which specifies the two methods `Install` and `Uninstall`.

---

##### Install and Uninstall

`Install` is called when the application is created in TAF. The parameters are used by TAF and should not be modified. It is recommended to add code for further initiations and allocation of system resources in this method, for example load assemblies, open communication channels, and so on.

When you close the application `Uninstall` is called. This happens right before the application is deleted. It is recommended to add code for disposal of system resources in this method, for example unload assemblies, close sockets, and so on.

---

##### ITpsViewActivation

All custom applications should implement the `ITpsViewActivation` interface, which specifies the two methods `Activate` and `Deactivate`. These are used by TAF to notify applications when they get focus and when they lose it.

---

##### Activate and Deactivate

`Activate` is run every time the application gets focus. This happens at initialization, after the `ITpsViewSetUp.Install` method has been run, and when you click the application icon in the task bar.

`Deactivate`, accordingly, is run every time the application loses focus. This happens when you close the application, before the `ITpsViewSetUp.Uninstall` method has been run, and when you click another application icon in the task bar.

##### **NOTE!**

It is recommended to activate and deactivate timers in these methods. This way timers will not run when other applications are in focus, thus saving processor power. For the same reason, any subscription to controller events should be disabled in `Deactivate` and enabled again in `Activate`. Note that current values should be read before enabling the subscription.



## 3 Run-time environment

### 3.2.6. ITpsViewSetup and ITpsViewActivation

*Continued*

#### Simple code examples

This table shows the basic things that the ITpsView methods are used for in a custom application:

Method	Usage
<b>Install</b>	Create the Controller object: VB: <code>AController = New Controller</code> C#: <code>aController = new Controller();</code>
<b>Activate</b>	Add subscription to controller event: VB: <code>AddHandler AController.OperatingModeChanged, AddressOf UpdateUI</code> C#: <code>AController.OperatingModeChanged += new OperatingModeChangedEventHandler(UpdateUI);</code>
<b>Deactivate</b>	Remove subscription to controller event: VB: <code>RemoveHandler AController.OperatingModeChanged, AddressOf UpdateUI</code> C#: <code>AController.OperatingModeChanged -= new OperatingModeChangedEventHandler(UpdateUI);</code>
<b>Uninstall</b>	Remove the controller object: VB: <code>If Not AController Is Nothing Then AController.Dispose() AController = Nothing End If</code> C#: <code>if (aController != null) { aController.Dispose(); aController = null; }</code>

### 3.3. Release upgrades and compatibility

---

#### About this section

This section addresses compatibility issues with different version of Robotware.

---

#### Matching FlexPendant SDK and RobotWare release

You should be aware that the FlexPendant SDKs 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 FlexPendant SDK 5.11.01 will be compatible with FlexPendant 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.

As for a FlexPendant SDK application this means that the runtime will change, that is, the FlexPendant SDK assemblies located in RobotWare will be different from the ones that were used when the application was developed. Generally, the only way to be sure that the existing application will work with a newer RobotWare version is to compile the source code with the FlexPendant SDK version that matches the intended RobotWare version.



#### NOTE!

You find all the details about compatibility between different FlexPendant 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 FlexPendant SDK release.

---

#### Prepared for change

To sum up, 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\(VisualStudio.71\).aspx](http://msdn2.microsoft.com/en-us/library/aa309387(VisualStudio.71).aspx)

### 3 Run-time environment

---

#### 3.3. Release upgrades and compatibility

## 4 Developing Controller applications

### 4.1. Introduction

#### About this chapter

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

It also discusses some specific programming issues that are important for FlexPendant 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 FlexPendant SDK class libraries, as these topics are covered in dedicated chapters.

#### Basic approach

In most aspects, using the FlexPendant 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.



#### NOTE!

When using the .Net class libraries for FlexPendant development, see *Version Information* to be sure that the class or method is supported on .NET Compact Framework 2.0, which runs on the FlexPendant.

Use this procedure to program the FlexPendant SDK application for the first time:

Step	Action
1.	<p><b>Before you start</b></p> <p>Learn the basics about FlexPendant SDK programming by reading all relevant sections of this manual. Feel reassured that this is a timesaving activity and do not rush into coding.</p> <p>If you are going to develop an application for the FlexPendant, a device with limited resources compared to a PC, studying the chapter <a href="#">Robust FlexPendant applications on page 165</a> is crucial.</p>
2.	<p><b>During development</b></p> <p>Frequently test application functionality.</p> <p>Do not wait too long before you test a FlexPendant application on a real FlexPendant device.</p>
3.	<p><b>After development</b></p> <p>If you develop a FlexPendant application, set up a long term test before use in production. If not, you risk that the FlexPendant slowly runs completely out of memory and crashes due to your application.</p> <p>Verify application behavior, performance and memory consumption. Use the services offered by ABB. For more information, see <a href="#">FlexPendant - Debugging and troubleshooting on page 179</a>.</p>

## 4 Developing Controller applications

---

### 4.2. Analysis and design

## 4.2. Analysis and design

---

### About this section

The purpose of FlexPendant 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. FlexPendant 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.

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 FlexPendant 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 FlexPendant 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 FlexPendant SDK. Do make sure that you have really understood what the end-users of your application need.





#### 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 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. <b>Note!</b> You are also recommended to read through the chapter <a href="#">Robust FlexPendant applications on page 165</a> before starting coding.

#### As complex or as easy as you wish

A simple custom application can be created in a day or two using FlexPendant 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 FlexPendant SDK programming.

## 4 Developing Controller applications

---

### 4.3. Controller events and threads

### 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 FlexPendant 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

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

The table shows some events that exist in the FlexPendant 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 both on the FlexPendant and 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 on how this is done along with code examples, see [Invoke method on page 51](#).

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 FlexPendant touch screen then stops responding and the application has to be restarted.

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

*Continues on next page*

**Invoke method**

All FlexPendant 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(AddressOf UpdateUI), New Object()
    {sender, e})
```

C#:

```
this.Invoke(new EventHandler(UpdateUI), new Object[] {sender, 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 in the following example:

VB:

```
Private Sub UpdateUI(ByVal sender As Object, ByVal e As
    System.EventArgs)
    Dim Args As ExecutionStatusChangedEventArgs
    Args = DirectCast(e, ExecutionStatusChangedEventArgs)
    Me.Label1.Text = e.NewStatus.ToString()
End Sub
```

C#:

```
private void UpdateUI(object sender, System.EventArgs e)
{
    ExecutionStatusChangedEventArgs args;
    args = (ExecutionStatusChangedEventArgs) e;
    this.label1.Text = e.NewStatus.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).

**TIP!**

If you are using the FlexPendant SDK there is further information about threads in [Thread affinity on page 176](#) and [Invoke on page 176](#).



## 4 Developing Controller applications

---

### 4.4. User Authorization System

#### 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 the FlexPendant or RobotStudio. If the controller connection for any reason is lost, you have 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

Before sensitive controller operations are performed, a FlexPendant SDK application should check that you are currently logged on and have the corresponding UAS rights.

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

VB:

```
Dim UAS As UserAuthorizationSystem =  
    Me.AController.AuthorizationSystem
```

C#:

```
UserAuthorizationSystem uas =  
    this.aController.AuthorizationSystem;
```

---

##### 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 necessary grants 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.ModifyRapidProgram) Then  
    aTask.LoadModuleFromFile(localFile, RapidLoadMode.Replace)  
End If
```

C#:

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



##### NOTE!

The FlexPendant 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.

---

#### MessageBox feedback

If a UAS grant is missing you should get information about it. This can be done in a message as shown in the following example:

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

VB:

```
GTPUMessageBox.Show(Me, Nothing, msg, title,  
System.Windows.Forms.MessageBoxIcon.Asterisk,  
System.Windows.Forms.MessageBoxButtons.OK)
```

C#:

```
GTPUMessageBox.Show(this, null, msg, title,  
System.Windows.Forms.MessageBoxIcon.Asterisk,  
System.Windows.Forms.MessageBoxButtons.OK);
```

---

#### 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 `FlexPendant SDK` throws a `UasRejectException` and the `PC SDK` throws a `GrantDemandRejectedException`.



#### **TIP!**

For more information on UAS and `Grant` members, see *Reference Manual FlexPendant SDK*.

## 4 Developing Controller applications

### 4.5. Exception handling

## 4.5. Exception handling

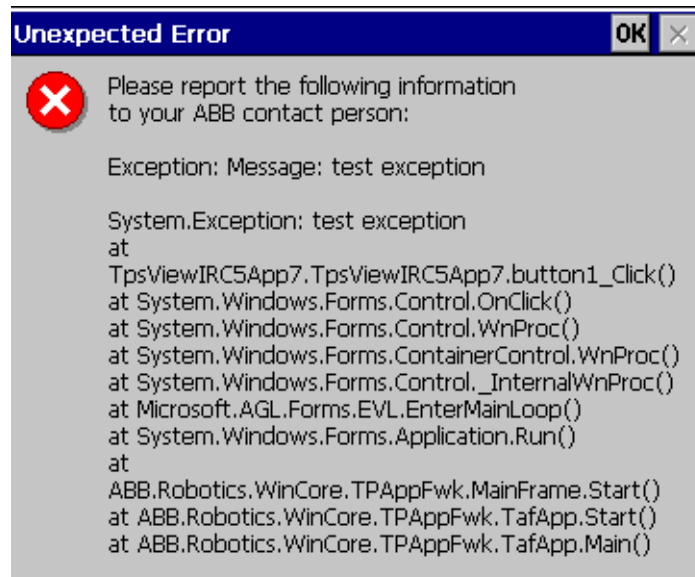
### Overview

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.

The FlexPendant SDK provides several exception classes, which are used when errors occur. If the operation is rejected by the controller safety access restriction mechanism, for example, a `RejectException` is thrown.

If an unhandled exception occurs the application crashes and TAF displays a gray message on the FlexPendant. After confirmation the FlexPendant will restart. To avoid this you should make sure that any potential error situations are dealt with properly by your code.



5.5\_1

The message that you get when an unhandled error situation occurs may look like this. Do NOT contact ABB, but fix the error handling of your application.



### NOTE!

For more information on how the exception classes of the FlexPendant SDK work, see *Reference Manual FlexPendant SDK*. Also see [SDK exception classes on page 174](#) to get more detailed information about exception handling for the FlexPendant platform.

---

#### 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, they can be caught and nothing need to be performed. 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`. For more information, see [Debug output on page 183](#) and [Trace and Debug on page 185](#).

---

#### 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)
    .....
```

C#:

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

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");
}
```

## 4 Developing Controller applications

---

### 4.5. Exception handling

*Continued*

---

#### **.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.



## 4.6. How to use the online help

### Overview

The online help comes with the installation of FlexPendant SDK and is accessible from Windows **Start** menu.

The recommendation is to read this application manual carefully while developing FlexPendant SDK applications. *Reference Manual FlexPendant SDK* is an important complements to this manual, as these make up the complete reference to the class libraries of FlexPendant SDK. For more information, see [Documentation and help on page 15](#).



#### **NOTE!**

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



#### **TIP!**

For more information on the web address to *RobotStudio Community*, where FlexPendant SDK developers discuss software problems and solutions online, see [Documentation and help on page 15](#).

## 4 Developing Controller applications

---

### 4.6. How to use the online help

# 5 Using the FlexPendant SDK

## 5.1 Introduction

### 5.1.1. About this chapter

---

#### Overview

This chapter gives detailed information on how to use the FlexPendant SDK.

These topics are covered:

- How to take advantage of some system features that support the use of customized screens.
- How to utilize the integrated Visual Studio wizard to set up a FlexPendant project.
- How to add the FlexPendant SDK GUI controls to the Visual Studio Toolbox.
- How to build the user interface using the integrated design support.
- How to program FlexPendant SDK GUI controls.
- How to launch other applications from your application.
- How to implement controller functionality using CAPI.

The design support in Visual Studio enables you to visually lay out the application, reducing the need to write code. This speeds up development and gives you a more precise control of the appearance of your FlexPendant screens.

Using the FlexPendant SDK it is possible to launch several of the standard FlexPendant applications from your application, which is often a very handy alternative to handling complicated procedures on your own, such as reading and writing RAPID data for example.

The Controller API (CAPI) is at the core of the FlexPendant SDK. It is used to access the robot controller, which the FlexPendant is attached to. First there is information about how this public API is organized. Then each domain of CAPI is dealt with separately. There are code samples in C# and VB throughout the chapter.

## 5 Using the FlexPendant SDK

### 5.1.2. System features supporting the use of customized screens

#### 5.1.2. System features supporting the use of customized screens

##### Flexible user interfaces

The FlexPendant can be adapted to end-users' specific needs in many different ways. It can be operated in 14 different languages, including Asian character-based languages such as Japanese and Chinese. Left-handed operators can adapt the device from its default setting by simply rotating the display through 180 degrees. Four of the eight hard keys are programmable, that is, their function can be assigned by the end-user.

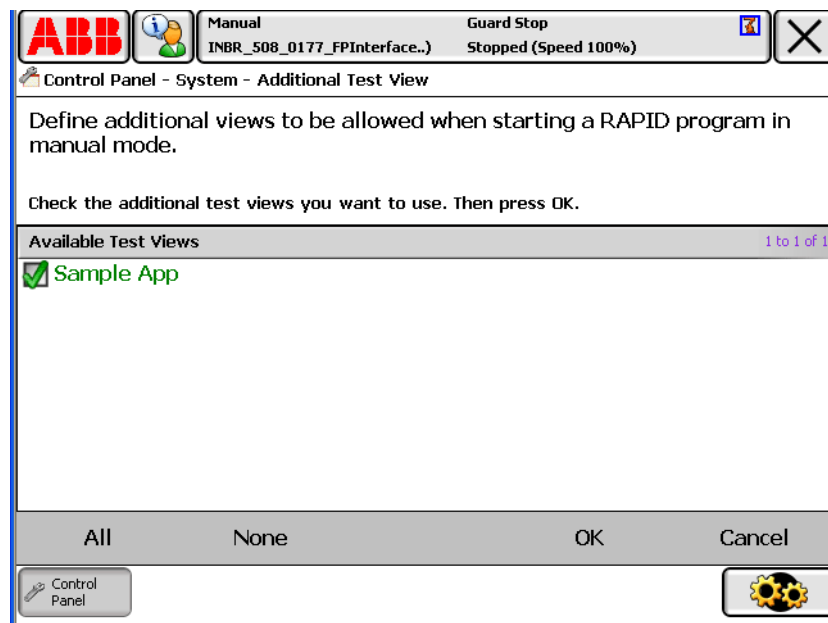
Customized FlexPendant screens, tailored to end-users' needs is yet another way of rendering the flexible solutions required by many customers. To support the use of customized screens there are a couple of features that you may want to tell the end-users of your application about.

##### Configure the FlexPendant

Using the FlexPendant configuration facilities (Control Panel - FlexPendant) it is possible to configure the FlexPendant to allow RAPID execution in manual mode from an FlexPendant SDK view. You can also make the FlexPendant automatically display an SDK view at operating mode change.

##### Additional Test View

Set the FlexPendant configuration property *Additional Test View* if you want to start RAPID execution in manual mode with a custom application as the active view.

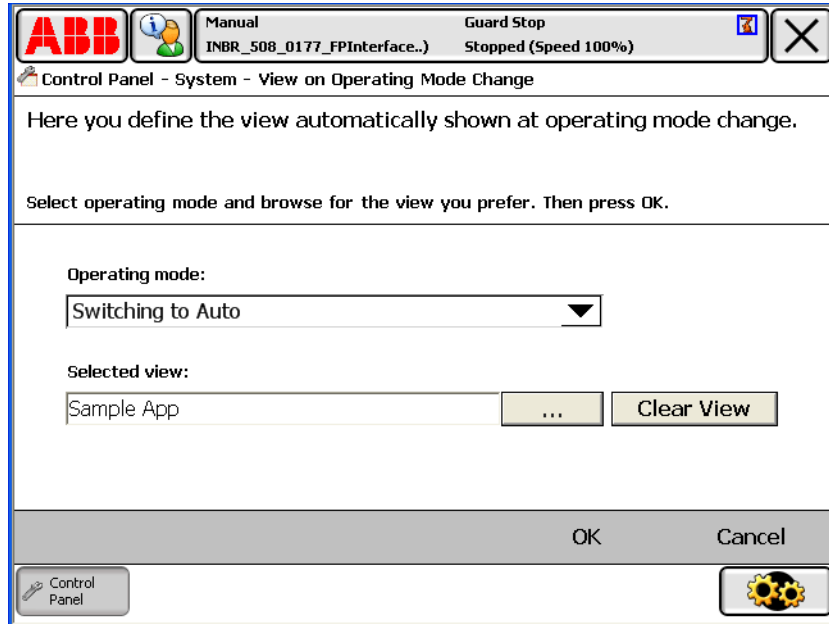


6.1.0\_1

Continued

## View On Operating Mode Change

Set the FlexPendant configuration property *View On Operating Mode Change* if you want the custom application to become active when the controller operating mode is switched to auto, manual or manual full speed.



6.1.0\_2

## Use RAPID instruction to launch FlexPendant SDK application

The RAPID instruction `UIShow` (User Interface Show) is used to communicate with the user of the robot system via a FlexPendant application. Both FlexPendant SDK applications and standard applications can be launched using this instruction.

Example:

The following RAPID code launches the custom application *TpsViewMyApp*.

```
CONST string Name := "TpsViewMyApp.gtpu.dll"; CONST string Type
:= "ABB.Robotics.SDK.Views.TpsViewMyApp"; UIShow Name, Type;
```

For this to work the robot system must have the RobotWare option *FlexPendant Interface*. The assemblies *TpsViewMyApp.dll* and *TpsViewMyApp.gtpu.dll* must be located in the HOME directory of the active robot system. (When the assemblies have been downloaded to the controller the FlexPendant must be restarted in order to load them.)

**NOTE!**

For more information on `UIShow` instruction, see *Technical reference manual - RAPID instructions, functions and data types*



## 5 Using the FlexPendant SDK

### 5.2.1. Using the project template in Visual Studio

## 5.2 Setting up a new project

### 5.2.1. Using the project template in Visual Studio

#### Overview

It is very simple to set up a new FlexPendant project using the integrated project template. It will automatically add the most common SDK references to the project and auto-generate some source code for the main application window, in the selected programming language, either C# or Visual Basic.



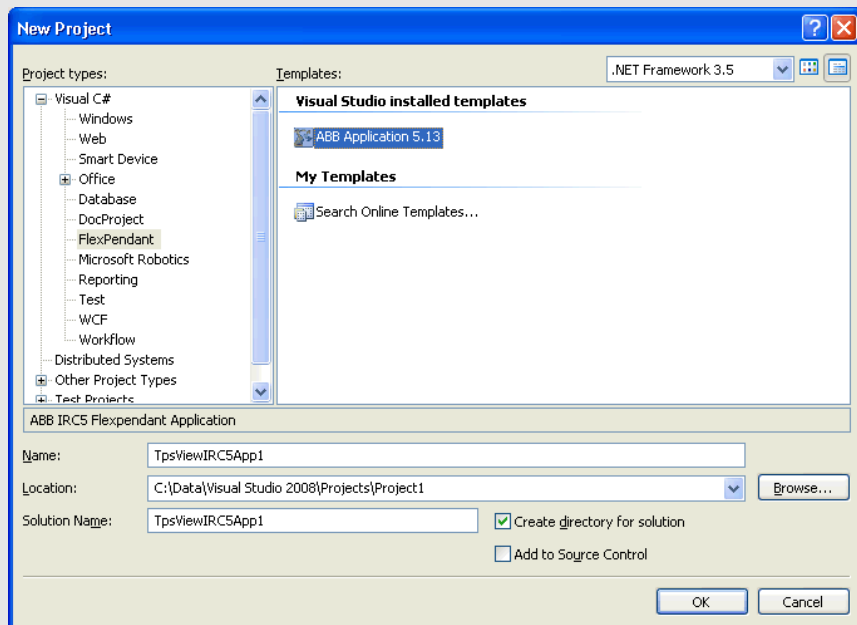
#### NOTE!

To add another view to the FlexPendant project, you do not need to start a new project. For more information, see [Adding a view to a custom application on page 83](#).

#### Setup procedure

Follow these steps to create a FlexPendant project:

Step	Action
1.	On the <b>File</b> menu in Visual Studio, point to <b>New</b> and then click <b>Project</b> .
2.	In Visual Studio 2005, in the <b>New Project</b> dialog select <i>Visual C# / Smart Device / FlexPendant</i> or <i>Other Languages/Visual Basic/Smart Device/FlexPendant</i> . In Visual Studio 2008, select <i>Visual C# / FlexPendant</i> or <i>Other Languages/Visual Basic/FlexPendant</i> ).





6.1.1\_1



#### NOTE!

If you have several FlexPendant SDK installations on your PC, there will be several templates to choose from. Make sure you select the template that match the RobotWare version that your application should target.

*Continued*

Step	Action
3.	<p>Enter the name of the application in the <b>Name</b> field and where you want it to be stored in the <b>Location</b> field. Click <b>OK</b>.</p>  <p><b>NOTE!</b> The name has to start by “<i>TpsView</i>”. If you forget it an error will be generated by the ABB verification tool when the project is built.</p>
4.	<p>The <b>FlexPendant SDK Project Wizard</b> is now launched. For information about how to configure the application using this wizard see <a href="#">FlexPendant TpsView attribute on page 39</a> and <a href="#">Container style on page 81</a>. When you are ready click <b>OK</b>.</p>
5.	<p>You need to set up the design support for the FlexPendant GUI controls before starting programming. How this is done is detailed in <a href="#">Setting up design support for FlexPendant controls on page 65</a>.</p>  <p><b>NOTE!</b> If the SDK references seem to be missing in your FlexPendant project, see the following section for instructions about how to solve the problem.</p>

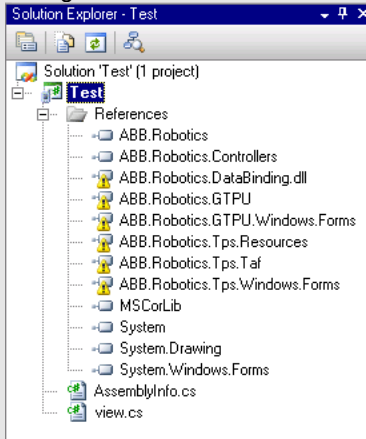

## 5 Using the FlexPendant SDK

### 5.2.1. Using the project template in Visual Studio

Continued

#### Add any missing references

The SDK references needed by the FlexPendant project are added automatically when the Project wizard is completed. However, there is a possibility that Visual Studio cannot provide the path to these dlls. If this is the case you should add the references path manually. Follow these steps:

Step	Action
1.	<p>Look at the <b>Solution Explorer</b> of Visual Studio. If you have a C# project expand the <b>References</b> node. Warning triangles mean that the path to the referenced dlls is missing.</p>  <p>6.1.1_2</p> <p>For a VB project the <b>Solution Explorer</b> looks the same, except that there is no <b>References</b> node.</p>
2.	<p><b>C#:</b> Select the project icon, right-click and select <b>Properties</b>. Click the <b>Reference Paths</b> tab and add the path to the SDK assemblies.</p> <p><b>VB:</b> Select the project icon, right-click and select <b>Properties</b>. Click the <b>References</b> tab. If the path to the SDK references is missing add it by browsing to the directory where they are located.</p>  <p><b>NOTE!</b></p> <p>The default path is <i>C:\Program Files\ABB Industrial IT\Robotics IT\SDK\FlexPendant SDK 5.xx\bin</i>.</p>
3.	<p>Save the project. You will notice that any warning triangles in the <b>Solution Explorer References</b> node will disappear.</p>



## 5.2.2. Setting up design support for FlexPendant controls

### Overview

This section describes how to make the FlexPendant GUI controls accessible in Visual Studio.

### Procedure

Follow these steps to add the FlexPendant controls to the Visual Studio **Toolbox**:

Step	Action
1.	On the <b>View</b> menu, select <b>Toolbox</b> .
2.	Right click in the <b>Toolbox</b> area and select <b>Add Tab</b> .
3.	Name the new toolbox tab, for example <i>FlexPendant Controls</i> .
4.	Right click in the area of the new tab and select <b>Choose Items</b> .
5.	In the <b>Choose Toolbox Items</b> dialog, browse to the directory where the FlexPendant SDK assemblies are located and import the following assemblies: <ul style="list-style-type: none"> <li>• ABB.Robotics.Tps.Windows.Forms.dll</li> <li>• ABB.Robotics.GTPU.Windows.Forms.dll</li> <li>• ABB.Robotics.DataBinding.dll</li> </ul> <p>The default location is <i>C:\Program Files\ABB Industrial IT\Robotics IT\SDK\FlexPendant SDK 5.xx\bin</i>.</p>
6.	In the Solution Explorer right-click view.cs (view.vb if you have a VB project) and select <b>View Designer</b> if you are not already in design mode. As you see, the FlexPendant specific controls are now accessible in the <b>Toolbox</b> . For more information on how to use them, see <a href="#">Introduction to visual design support on page 66</a> .



### NOTE!

The way you use the Visual Studio Designer to implement FlexPendant controls is very similar to implementing ordinary .NET controls. In this manual useful information which may not be obvious for all users is provided. But oftentimes, it is the general Visual Studio Help that will answer any questions you may have about control properties and the like.

## 5.3 Building the user interface

### 5.3.1. Introduction to visual design support

---

#### What is visual design support?

*Design Support for Compact Framework User Controls* was a new feature of Visual Studio 2005. It enabled design support for FlexPendant SDK controls to be included in FlexPendant SDK 5.08.

From FlexPendant SDK 5.08 onwards you visually design the FlexPendant application user interface in the Visual Studio Designer. FlexPendant controls are dragged from the toolbox to the designer area, moved and resized by clicking and dragging. By applying different settings in the **Properties** window of a control, you refine its appearance and behavior.

To use the visual design support you must add the FlexPendant controls to the Visual Studio toolbox. How to do this is detailed in [Setting up design support for FlexPendant controls on page 65](#).



#### NOTE!

Design support for FlexPendant controls has long been on FlexPendant SDK users' wish list. It is indeed a time-saving feature, as most of the code supporting the graphical user interface is now auto generated.

---

#### Why special controls for the FlexPendant?

You may wonder why the standard Microsoft Windows controls have not been considered good enough to be used for the FlexPendant touch screen.

The answer is that some Windows controls may very well be used. Other Windows controls are however not so well suited for the FlexPendant touch screen. To navigate using your finger controls in particular need to be large enough. In some other cases the Windows controls simply do not look very good on the touch screen.



#### NOTE!

In the *Reference Manual FlexPendant SDK*, click the **Contents** tab and the **ABB.Robotics.Tps.Windows.Forms** node to get an overview and a short description of all ABB controls you may use to create the user interface of a FlexPendant application.

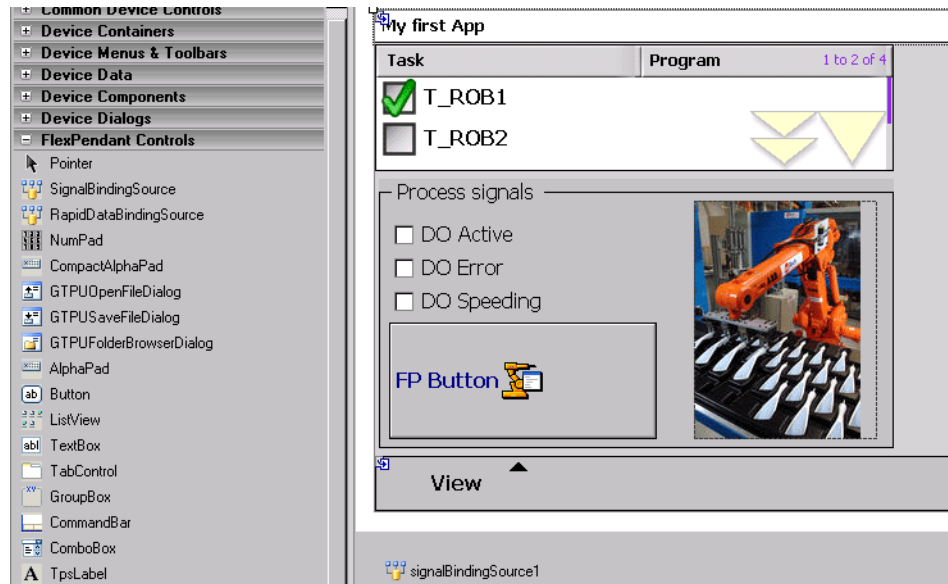
#### TIP!

How to program these ABB controls is very similar to the equivalent Windows controls. If you need code examples the best source of information is usually MSDN. You may try <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlfsystemwindowsformslistviewclasstopic.asp>, for example, to find out how to program a `ListView`.



**Illustration**

The following screenshot shows the Visual Studio **Toolbox** with all of the FlexPendant controls to the left. In the **Designer** area to the right, a FlexPendant application is being developed. Part of the container control, along with a number of ABB and standard Windows controls can be seen.



6.3.1\_1

The following features are worth noting:

- A Windows **PictureBox** control is used as the container of a photo.
- The FlexPendant button has a property called **BackgroundImage**, used to display an image. The ABB image library located in the *Resources* folder, which comes with the installation of FlexPendant SDK, has numerous icons, which can be used by custom applications. You can of course also use photos and icons of your own.
- Some of the ABB controls, such as the **SignalBindingSource**, have no graphical representation in design-time. As you see in the figure, they are placed in the components pane under the main form. Code is generated similar to the controls that you see on the form.
- Usually a mix of Windows and ABB controls are used for a FlexPendant SDK application. For example, as there is no ABB **RadioButton** or **CheckBox** the equivalent Windows controls are used. In the figure, the standard Windows **CheckBox** is used.

**CAUTION!**

Do not forget about the limited amount of memory available for custom applications when adding images or photos! For more information, see [Technical overview of the FlexPendant device on page 165](#).

**CAUTION!**

Auto generated code for controls is located in the method `InitializeComponent`. You should not tamper with the code inside this method. Any modifications or additions to auto generated code is usually best located in the constructor, after the call to `InitializeComponent`.

Continues on next page



## 5 Using the FlexPendant SDK

---

### 5.3.1. Introduction to visual design support

*Continued*


---

#### Hands on - Hello world

Are you ready to program and test your first FlexPendant application? If you have not created a FlexPendant project and added the FlexPendant controls to the Visual Studio Toolbox you need to do that first. For more information, see [Using the project template in Visual Studio on page 62](#) and [Setting up design support for FlexPendant controls on page 65](#).

Follow these steps to create and test a simple custom application:

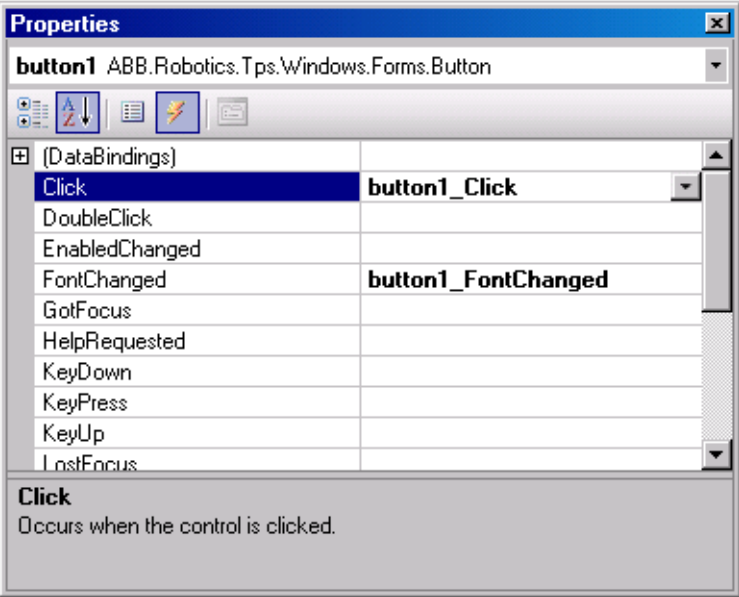

Step	Action
1.	Drag a FlexPendant button from the <b>Toolbox</b> to the <b>Designer</b> .
2.	Double-click the button in the <b>Designer</b> , this opens the code editor.

Step	Action
3.	<p>As you see an event handler for the click event has been created. Add the code to launch the Hello World message in it:</p> <pre data-bbox="587 389 1433 1503"> private void button1_Click(object sender, EventArgs e) {     try     {         myLabel1.Text = " Hello World! ";     }     catch(System.Exception ex)     {         // Catch is mandatory here!!         // If there is no catch to take care of an error         // caused by your application,         // not only your application, but the entire         // FlexPendant, will crash, as all FP applications         // share the same GUI-thread.         // It is up to you how to handle the error. Normally,         // you may want to display an error message,         // something like this:         GTPUMessageBox.Show(this.Parent         , null         , "An unexpected error occurred \nError Message: "         +ex.Message         , "Application Error"         , System.Windows.Forms.MessageBoxIcon.Question         , System.Windows.Forms.MessageBoxButtons.OK);     }     finally     {         // cleanup code goes here (if needed)         // "finally" is used to guarantee that cleaning up         // resources, releasing mastersip etc is not         // skipped due to the exception.         // (In this case, the finally clause is not         // necessary, as there is nothing to clean up.)     } } </pre> <p>An alternative way of adding an event handler is shown in the next step.</p> <div data-bbox="539 1592 619 1666" style="text-align: center;">  </div> <p><b>NOTE!</b></p> <p>A FlexPendant application to be used in industry must have exception handling in all event handlers, as shown in the preceding code sample. The FlexPendant has only ONE GUI thread, which all applications running on the FlexPendant share. If your application breaks the only GUI thread, all of the applications will die and the FlexPendant must be restarted manually. For more information, see <a href="#">Exception handling on page 54</a>.</p>

## 5 Using the FlexPendant SDK

### 5.3.1. Introduction to visual design support

Continued

Step	Action
4.	<p>Bring up the <b>Properties</b> window for the button by right clicking it in the <b>Designer</b> and selecting <b>Properties</b>. Click the events button (yellow flashing lightning) to see the events available for the ABB button. As shown by the figure, you have already connected an event handler to the click event.</p>  <p>6.3.1_1B</p> <p> <b>NOTE!</b></p> <p>It is the default event of the control that you get when you double-click it in the Designer. To add an event handler for another button event, bring up the Properties window and double-click the event you want to generate a handler for, for example a <code>FontChanged</code> event handler. You will enter the code editor, the cursor inside the generated event handler.</p>
5.	<p>On the <b>Build</b> menu, click <b>Build Solution</b> and check that you did not get any compilation errors.</p>
6.	<p>To test the application you need to deploy it to a robot system. If there is no robot system available, you must create a robot system by using the <i>System Builder</i> of RobotStudio.</p>
7.	<p>Copy the assembly (*.dll) and the proxy assembly (*.gtpu.dll) from the bin\Debug (or bin\Release) directory of your Visual Studio project to the HOME directory created by RobotStudio when your robot system was created.</p> <p>Example showing default paths for copy/paste: C:\Data\Visual Studio 2005\Projects\TpsViewMyApp\TpsViewMyApp\bin\Debug\ C:\Systems\sys1_508\HOME</p> <p><b>Note!</b> If you have already started the virtual FlexPendant you need to close it before pasting the dlls.</p>
8.	<p>Start the virtual FlexPendant.</p>
9.	<p>On the ABB menu, find your application and open it.</p>

© Copyright 2010 ABB. All rights reserved.

Continues on next page

Step	Action
10.	Click the button. The <i>Hello World</i> message will be displayed.

6.3.1\_2

**NOTE!**

For more information on how to implement the additional features: application title, button background image, command bar with a menu item and an event handler to launch a standard view, see [Hands on - step 2 on page 72](#).



**TIP!**

When using the Virtual FlexPendant for test and debug you can automate the copy/paste procedure described in the preceding step seven. The following procedure explains it:

1. Right click the project icon in the **Solution Explorer** and select **Properties**.
2. For a C# project, select the **Build Events** tab. (For a VB project click the **Compile** tab and then the **Build Events** button.)
3. Click the **Edit Post-build** button and add two commands which will copy the produced dlls to the directory of the robot system to run. Example:

```
copy "$ (TargetDir) $ (TargetName) .dll" "C:\Systems\sys1"
copy "$ (TargetDir) $ (TargetName) .gtpu.dll" "C:\Systems\sys1"
```

**Note!** Do not remove the default post build command lines, which create the assembly proxy (\*gtpu.dll).

## 5 Using the FlexPendant SDK


### 5.3.1. Introduction to visual design support

*Continued*

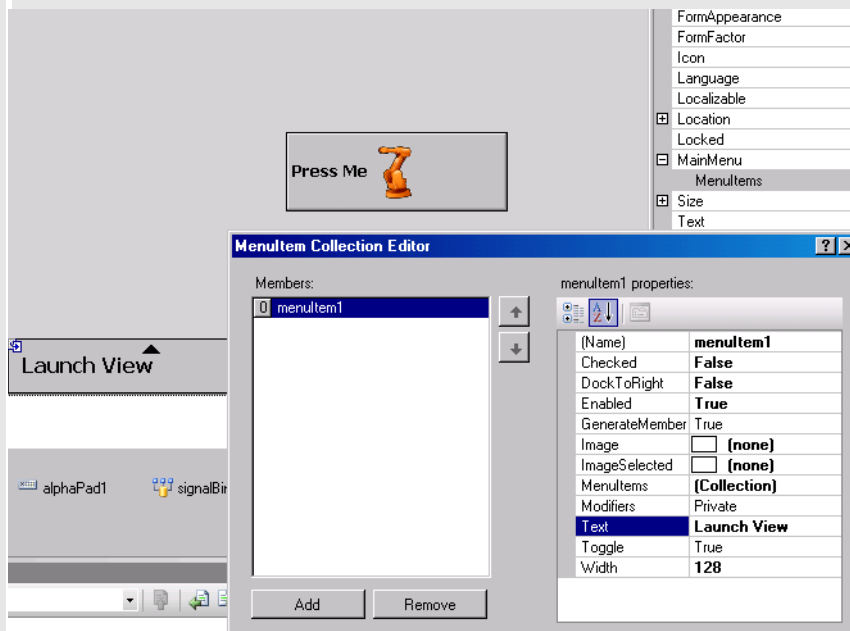

#### Hands on - step 2

This section details how to program the remaining features of the *Hello World* application shown in preceding step nine: application title, button background image, command bar with a menu item. You will also learn how to implement an event handler for the **Launch View** command.

When you create a FlexPendant project you choose **Empty** or **Form** in the **FlexPendant SDK Project Wizard**. For more information to understand the difference between them, see [Empty or Form? on page 82](#). In our example **Form** is selected as the container control.

Step	Action
1.	<p>If you selected <b>Empty</b> for your project you can change it to <b>Form</b> by exchanging <code>TpsControl</code> with <code>TpsForm</code> directly in the source code, like this:</p> <p><b>C#:</b></p> <pre>public class TpsViewIRC5App14 : TpsForm, ITpsViewSetup, ITpsViewActivation</pre> <p><b>VB:</b></p> <pre>Public Class TpsViewIRC5App14 Inherits TpsForm '(was TpsControl before) Implements ITpsViewSetup, ITpsViewActivation</pre>  <p><b>NOTE!</b> The <b>Designer</b> now shows the title bar on top of the container control.</p>
2.	Right-click somewhere in the middle of the container control and select <b>Properties</b> .
3.	Enter "This is my Hello World application" at the <b>Text</b> property. See the tip of <a href="#">Empty or Form? on page 82</a> if the <b>Text</b> property is not available.
4.	Still using the <b>TpsForm Properties</b> window, expand the <b>MainMenu</b> property node.
5.	Click the <b>MenuItems</b> property and the browse button which appears once the property is selected.




Step	Action
6.	<p>In the <b>Menulitem Collection Editor</b>, which is now launched, click the <b>Add</b> button. Then enter "Launch View" at the <b>Text</b> property. Click <b>OK</b>.</p>  <p>6.3.1_4</p> <p> <b>NOTE!</b></p> <p>The added menulitem has its own <b>Menulitems</b> property. You use it if the "Launch View" command is to present a menu instead of working as a button on the command bar. How this can be done is further detailed in <a href="#">How to add menu items on page 85</a>.</p>

## 5 Using the FlexPendant SDK

### 5.3.1. Introduction to visual design support

Continued

Step	Action
7.	<p>Add an event handler with code that will launch the standard <i>Jogging</i> view when the Launch View button is pressed.</p> <p>Adding event handlers is done differently in Visual Studio depending on which programming language you are using.</p> <p><b>For C#:</b></p> <p>In the constructor, after the call to <code>InitializeComponent</code>, add a subscription to the click event of the “Launch View” command. Write:</p> <pre>menuItem1.Click +=</pre> <p>then TAB twice. As you will notice, Visual Studio intellisense auto generates an event handler, and you only need to write the code for launching the Jogging view:</p> <pre>this._launchService.LaunchView(FpStandardView.Jogging,null,false,out this._cookieJog)</pre> <p><b>For VB:</b></p> <p>Add an event handler using the preceding drop down boxes the Visual Studio Code Viewer. Find and select <b>MenuItem1</b> in the left drop down box and the <b>Click</b> event in the right drop down box. The event handler is now auto generated for you, and the only thing you need to do is to write the code for launching the Jogging view (see VB code in <a href="#">Launching standard views on page 112</a>).</p>
	<p></p> <p><b>NOTE!</b></p> <p>There is no possibility to attach an event handler in the properties window, like you do for a <code>Button</code> for example.</p>
8.	<p>Declare the <code>_launchService</code> and the <code>_cookieJog</code> objects:</p> <pre>private ABB.Robotics.Tps.Taf.ITpsViewLaunchServices _launchService;private object _cookieJog;</pre>
9.	<p>Retrieve the <code>_launchService</code> object in the <code>Install</code> method of your class:</p> <pre>/// &lt;summary&gt; /// This method is called by TAF when the control is installed in the framework. /// &lt;/summary&gt; bool ITpsViewSetup.Install(object sender, object data) {     if (sender is ITpsViewLaunchServices)     {         // Save the sender object for later use         this._launchService = sender as ITpsViewLaunchServices;         return true;     }     return false; }</pre> <p>6.3.1_5</p> <p>For VB code example, see <a href="#">Installing ITpsViewSetup on page 111</a>.</p>
10.	<p>In the designer, open the <b>Properties</b> window of the button.</p>
11.	<p>Enter “Press Me” at the <b>Text</b> property.</p>
12.	<p>Select the <b>BackgroundImage</b> property and browse to the <i>Resources</i> folder of the FlexPendant SDK installation and import “IRB140.gif”.</p> <p>(Default path: <code>C:\Program Files\ABB Industrial IT\Robotics IT\SDK\FlexPendant SDK 5.08\Resources</code>)</p>
13.	<p>Build, deploy and start the application, that is repeat step five to eight in <a href="#">Hands on - Hello world on page 68</a>.</p>
14.	<p>Click the <b>Launch View</b> command.</p> <p>The Jogging view should open up and get focus.</p>

© Copyright 2010 ABB. All rights reserved.

Continues on next page

*Continued*



#### **TIP!**

For more information about the possibilities to use already existing views for your application, see *Using launch service on page 111* and *Using standard dialog box to modify data on page 114*.

---

### **Visual design and user experience**

There is another definition of the term *Visual design*, which is not in the scope of this manual. It has to do with how the design of the software user interface affects user experience. This is nonetheless an important topic for a FlexPendant application developer. Knowing what makes a user interface intuitive and easy to use is essential, as this is exactly what is expected from a custom operator interface.



#### **TIP!**

A FlexPendant style guide comes with the installation of FlexPendant SDK. It will help you to present application functionality in an intuitive way, teaching best practices of visual design and user experience from the FlexPendant perspective. It is a preliminary draft, but still very useful.

You may also want to study the Microsoft standard for visual design at:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwue/html/ch14a.asp>

### 5.3.2. GUI controls and memory management

---

#### Overview

All of the FlexPendant SDK controls belong to the `ABB.Robotics.Tps.Windows.Forms` namespace. This namespace thus includes well over 40 classes and subclasses used to develop the user interface of a FlexPendant application.

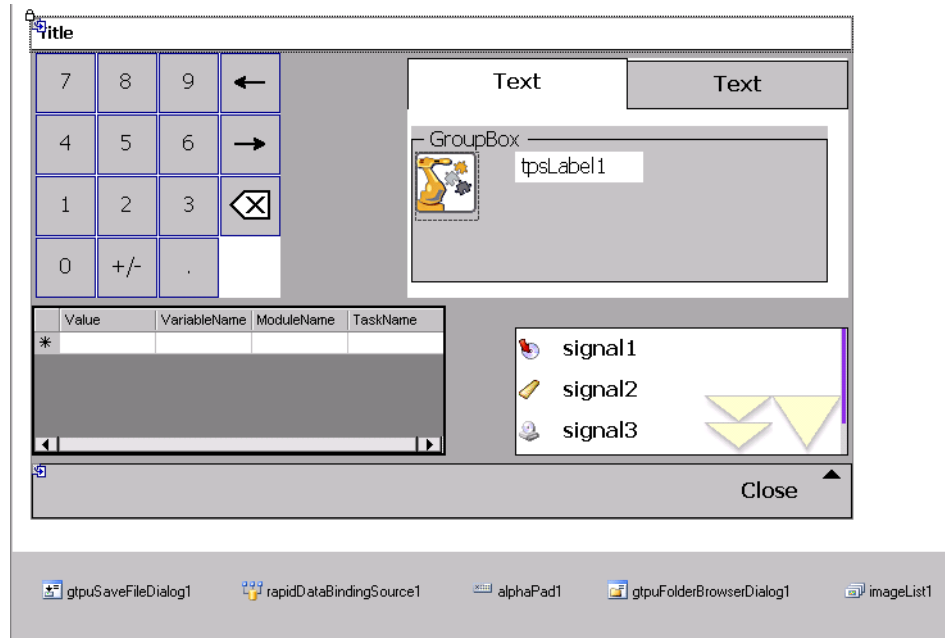
`ABB.Robotics.Tps.Windows.Forms.TpsControl` is the base type of all FlexPendant SDK controls. `TpsControl` in turn extends `System.Windows.Forms.UserControl`

`TpsControl` may be used as a container control of a FlexPendant view. It also has the default implementation of `Dispose`, which is called in order to free allocated memory for a control which should no longer be used.

You may wonder why this is necessary, as the .NET framework has a garbage collector, which should release the developer of the duty to free allocated memory. The answer is the garbage collector does not always reclaim memory which is no longer used. Therefore, if you program an application meant to execute around the clock on a device with limited memory, you are still responsible for freeing memory which is no longer used. Neglecting to do so will result in permanent memory leaks.

### How to avoid memory leaks

Look at this figure showing a number of controls (both ABB and Microsoft) and learn the basics about memory management for GUI controls. It is not so complicated, the most important thing is not to forget that cleaning up is your responsibility!



6.3.2\_1

Learn and apply these the general rules for memory management of GUI controls.

- Controls with a graphical representation, e.g the ABB Numpad, TabControl, GroupBox, TpsLabel, ListView and the Microsoft PictureBox and DataGrid in the figure, are automatically added to the controls collection in `InitializeComponent`. It may look like this:  

```
this.Controls.Add(this.numPad1);
```
- If the preceding figure represents the first view of your application, controls with graphical representation will be disposed of by the base class of your view class when your application is shut down and the `Dispose` method is called by TAF. This happens when the following statement in your `Dispose` method is executed:  

```
base.Dispose(disposing);
```
- If, however, it represents a secondary view of your application (which is actually the case here, as you can tell from the close button on the command bar), you must call its `Dispose` method from the first view when it is closed. Its base class will then remove all controls that are part of its controls collection, like in the previous case.
- GUI controls that have no graphical representation, but are located in the *Components pane* under the form, for example `GTPUSaveFileDialog`, `RapidDataBindingSource`, `AlphaPad` and so on are NOT added to the controls collection by default. These are the ones that you need to be especially careful to remove, as no garbage collector will ever gather them. If you forget to explicitly call `Dispose` on such controls you will have caused a permanent memory leak. Carefully study the code example in the next section.

Continues on next page

*Continued*



#### **NOTE!**

Microsoft and ABB controls behave in the same way. The Microsoft `ImageList` for example, which is commonly used in FlexPendant applications, has no graphical representation and must thus be explicitly removed by the application programmer.

---

#### **Coding the Dispose method**

The following code example shows how the `Dispose` method of the view shown in the preceding figure can be coded. All controls located in the components pane in the Designer must be explicitly removed by the programmer. Controls with a graphical representation will be removed when `Dispose` of the base class is executed.

```
protected override void Dispose(bool disposing)
{
    if (!IsDisposed)
    {
        try
        {
            if (disposing)
            {
                //Removes SaveFile dialog
                if(this.gtpuSaveFileDialog1 != null)
                {
                    this.gtpuSaveFileDialog1.Dispose();
                    this.gtpuSaveFileDialog1 = null;
                }
                //Removes RapidDataBindingSource
                if(this.rapidDataBindingSource1 != null)
                {
                    this.rapidDataBindingSource1.Dispose();
                    this.rapidDataBindingSource1 = null
                }
                //Removes Alphapad
                if(this.alphaPad1 != null)
                {
                    this.alphaPad1.Dispose();
                    this.alphaPad1 = null
                }
                //Removes FolderBrowserDialog
                if(this.gtpuFolderBrowserDialog1 != null)
                {
                    this.gtpuFolderBrowserDialog1.Dispose();
                    this.gtpuFolderBrowserDialog1 = null
                }
                //Removes ImageList
                if(this.imageList1 != null)
                {
                    this.imageList1.Dispose();
                    this.imageList1 = null
                }
            }
        }
    }
}
```

```

        }
    }
}
finally
{
    //Removes all controls added to the controls collection
    base.Dispose(disposing);
}
}
}

```

Finally, as this is a secondary view, we should call its `Dispose` method from the first view when it is closed down.

```

//This code is executed by the first view when the secondary view
is closed
void form2_Closed(object sender, EventArgs e)
{
    if(form2 != null)
    {
        form2.Dispose();
        form2 = null;
    }
}
}

```

**CAUTION!**

If you forget to call `Dispose` on controls that are not part of the control collection of the class there will be memory leaks. This may cause the FlexPendant to run completely out of memory and crash. Usually, this will not happen when you try out the functionality of your application, but when it is executed and used continuously during production. To verify that a GUI control is really disposed of, you may set up a subscription to its `Disposed` event for example, and verify that it is triggered when you close down the view.

**CAUTION!**

All objects accessing robot controller services, that is, unmanaged resources, must also be removed by the application programmer. For more information, see [Memory management on page 167](#).

*Continued*

---

#### Freeing allocated memory for a GUI control

You are recommended to remove a GUI control in the `Dispose` method of the class that created it. If the control belongs to the first view of your application, it will be disposed of when TAF calls `Dispose` at application shut down. If it belongs to a secondary view, you are responsible for disposing of the secondary view and its controls.

```
C#:  
if (this.controlX != null)  
{  
    controlX.Dispose();  
    controlX = null;  
}  
base.Dispose(disposing);  
  
VB:  
If disposing Then  
    If Not controlX Is Nothing Then  
        controlX.Dispose()  
        controlX = Nothing  
    End If  
End If  
MyBase.Dispose(disposing)
```



#### **NOTE!**

When the last statement in the preceding code example is executed the base class will call `Dispose` on all controls added to the controls collection in `InitializeComponent`. This means that you do not need to call `Dispose` on such controls.

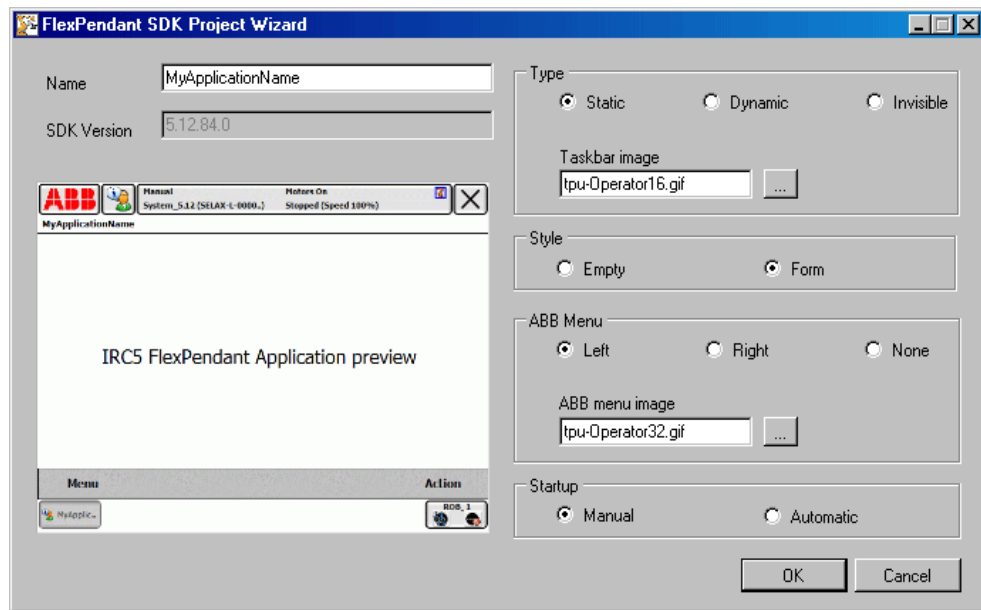


### 5.3.3. Container style

#### Overview

The container control of a FlexPendant application is 640x390 pixels, which is the exact size of the FlexPendant screen dedicated for a custom view. This section details the two different styles that can be used for a FlexPendant view: **Empty** and **Form**.

You select which one to use in the **Style** area of the **FlexPendant SDK Project Wizard**. To the left in the wizard, a preview helps you see the difference between the two styles.



6.2.2\_1

Here **Form** is selected. The preview illustration shows a `TpsForm`. It has a `CommandBar` at the bottom and a `TpsCtrlTitleBar` at the top.

#### NOTE!

When adding a new view to an existing FlexPendant project, you choose which control to use in the **Add New Item** dialog box. For more information on how to add a new view to an existing project, see [Adding a view to a custom application on page 83](#).



*Continued*

---

#### Empty or Form?

Your choice in the wizard determines the type that your view class will inherit. If you change your mind about this, it is easy to make the change directly in the auto generated code.

##### Empty

This is a plain `TpsControl`. It has neither a `CommandBar`, nor a `TpsCtrlTitleBar`.

The code definition of the class will look like this:

VB:

```
Public Class TpsViewMyApp Inherits TpsControl Implements  
ITpsViewSetup, ITpsViewActivation
```

##### Form

This is a `TpsForm`, which is a specialized `TpsControl`.

The code definition of the class will look like this:

VB:

```
Public Class TpsViewMyApp Inherits TpsForm Implements  
ITpsViewSetup, ITpsViewActivation
```



#### NOTE!

Often **Empty** is chosen as the first view of an FlexPendant SDK application. For any secondary view, which is to be opened from the first one, **Form** is the recommended container control.



#### NOTE!

When selecting **Form** you might wonder why the command bar is not visible in the designer. The answer is that it remains invisible until a `MenuItem` has been added to it. This is done in the **Properties** window of the `TpsForm`, by applying the property **MainMenu**.



#### TIP!

If you select the `TitlePanel` or the `CommandBar` of a `TpsForm` you will see that the **Properties** window is disabled. If you want to change the text of the title panel, or add menu items to the command bar, for example, you must select `TpsForm` by clicking somewhere in the middle and then modify its **Text** or **MainMenu** property.

---

### How to build the command bar

The command bar is either ready-made, as for `TpsForm`, or used as a separate control available from the toolbox. If the command bar is ready-made, the design of it is done by modifying the **MainMenu** property of `TpsForm`. If used as a separate control, you design it by modifying its own properties. Apart from this, there is no difference in how you control the design.

An important thing to remember is that you have to manually add the code controlling that will happen when you click a menu item, as there are no event properties available in its **Properties** window.

For more information on how to implement the control and its event handlers, see [Command bar on page 85](#).



### CAUTION!

The command bar of the *first* view of a custom application should not have a **Close** button. The reason is that all custom application must be closed down by TAF, which happens when you click the close [ x ] button in the upper right corner of the FlexPendant display. (For definition of TAF, see [Definitions on page 17](#) or for more information on TAF, see [Understanding the FlexPendant application life cycle on page 37](#).)

---

### Adding a view to a custom application

To add another view to the FlexPendant project, you do not need to start a new project. The following procedure explains how to proceed:

Step	Action
1	Right-click the project node in the <b>Solution Explorer</b> , point to <b>Add</b> and select <b>New Item</b> .
2	Select one of the FlexPendant container controls available in the dialog box. Normally it will be a <i>Form</i> . <b>Note!</b> If you select <b>Empty</b> the code you write to open it from the first view is a bit different than if you are using <b>Form</b> as a secondary view.

This way all of your views make up a single dll, which is convenient. Most of the time there is no point in dividing the application into several dlls.

### Launching the view

The code for launching the secondary view is simple. It may look like this:

```
//declare the secondary view as a private member
private View2 _view2;

//create the view and add subscription to its closed event, then launch
view

private void button1_Click(object sender, EventArgs e)
{
    this._view2 = new View2();
    this._view2.Closed += new EventHandler(view2_Closed);
    this._view2.ShowMe(this);
}
```

*Continues on next page*

### 5.3.3. Container style

*Continued*

```
}  
//dispose of the view when it has been closed  
void view2_Closed(object sender, EventArgs e)  
{  
    this._view2.Closed -= new EventHandler(view2_Closed);  
    this._view2.Dispose();  
    this._view2 = null;  
}
```



#### **NOTE!**

Make sure that there is a natural way for you to get back to the preceding view. The most intuitive way of doing this is to click a **Close** button or an **OK** or **Cancel** button on the command bar. If the secondary view is a `TpsForm`, this event handler closes it:

```
void menuItem1_Click(object sender, EventArgs e) { this.CloseMe();}
```



#### **NOTE!**

If you are using **Empty** as a secondary container control the code you write to launch it is:

```
this._view2.Show();
```

It must be added to the first view's control collection before it can be shown, like an ordinary .NET control.

### 5.3.4. Command bar

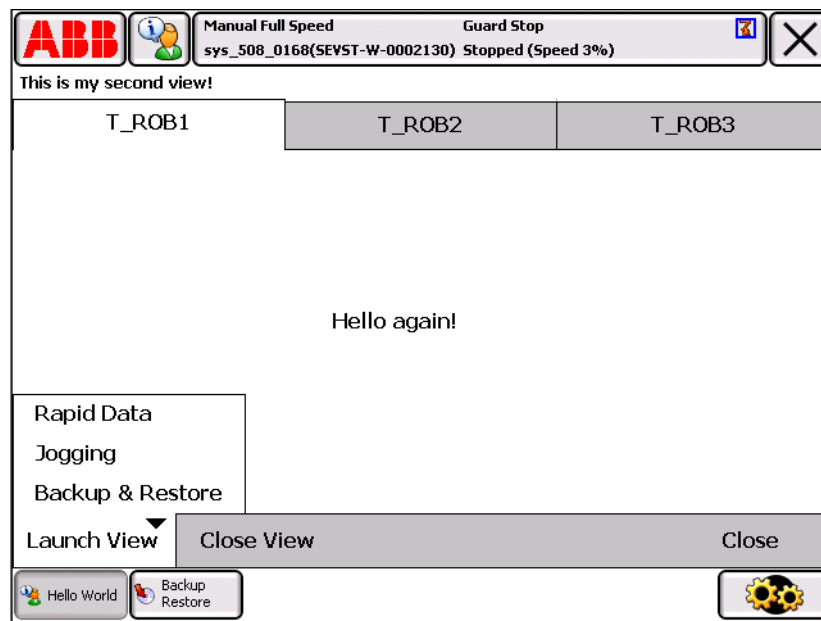
#### Overview

For the end-user the command bar is a central FlexPendant control and an important means of interacting with the system. As described in *How to build the command bar on page 83* it can be ready-made or added to the container control from the Visual Studio Toolbox.

#### How to add menu items

If you use the `Form` container a command bar is built-in, but not visible until you add a `MenuItem` collection to the `MainMenu` property of `TpsForm`. How to do this is shown in step 5 - 6 of *Hands on - step 2 on page 72*.

If you use `Empty` as container the command bar needs to be added from the Toolbox. In this case too, the commands are added as a collection of menu items.



6.3.4\_1

The command bar in this figure has a collection of three menu items: *Launch View*, *Close View* and *Close*. Moreover, the *Launch View* menu item has its own collection of menu items: *Rapid Data*, *Jogging* and *Backup Restore*.

In the figure, as shown by the task bar, the *Backup Restore* view has been opened. It has been done by clicking the *Backup Restore* menu item. The user has then returned to the **Hello World** application by using the task bar. If *Close View* is now pressed, the *Backup Restore* view will close down. The next section shows how event handlers for this command bar can be implemented.

*Continues on next page*

*Continued*

#### How to add menu item event handling

Subscriptions to menu item events and event handling cannot be added using the design support, but have to be implemented by hand in the source file. The following code example shows how the command bar in the preceding figure can be implemented.

```
private ABB.Robotics.Tps.Taf.ITpsViewLaunchServices _launchService;
private object _cookie;

//Constructor, takes ITpsViewLaunchServices object as in parameter
public View2(ABB.Robotics.Tps.Taf.ITpsViewLaunchServices IS)
{
    _launchService = IS;
    InitializeComponent();
    this.menuItem3.Click += new EventHandler(menuItem3_Click);
    this.menuItem4.Click += new EventHandler(menuItem4_Click);
    this.menuItem5.Click += new EventHandler(menuItem5_Click);
    this.menuItem6.Click += new EventHandler(menuItem6_Click);
    this.menuItem7.Click += new EventHandler(menuItem7_Click);
}

//Event handlers for all menu item commands
void menuItem3_Click(object sender, EventArgs e)
{
    //opens RapidData view, may open several instances as specified by 3:rd argument
    this._launchService.LaunchView(FpStandardView.RapidData, null, true, out this._cookie);
}

void menuItem4_Click(object sender, EventArgs e)
{
    //opens Jogging view
    this._launchService.LaunchView(FpStandardView.Jogging, null, false, out this._cookie);
}

void menuItem5_Click(object sender, EventArgs e)
{
    //opens BackUp & Restore view
    this._launchService.LaunchView(FpStandardView.BackUpRestore, null, false, out this._cookie);
}

void menuItem6_Click(object sender, EventArgs e)
{
    if (_cookie != null) //if this app. has opened a standard view, this code closes it
    {
        this._launchService.CloseView(this._cookie);
        this._cookie = null;
    }
}

void menuItem7_Click(object sender, EventArgs e)
{
    this.CloseMe(); //closes secondary view
}
}
6.3.4_2
```



#### NOTE!

To launch a standard view the `ITpsViewLaunchServices` object, which can be saved by the `TpsView` class in its `Install` method, is needed. For more information, see [Using launch service on page 111](#).

---

### 5.3.5. FlexPendant fonts

---

#### Overview

On the FlexPendant, `TpsFont` is used instead of standard Windows fonts. By using `TpsFont` you save limited device memory, as static references instead of new font instances will be used.

---

#### TpsFont

`Tahoma` is the default font of the FlexPendant. It is also the font usually used internally by `TpsFont`. A number of different font sizes are pre-allocated and can be reused. You are recommended to use `TpsFont` instead of creating new font instances for both ABB and .NET UI controls.

**NOTE!**

To use Chinese or another language with non-western characters, you must use the FlexPendant font, `TpsFont`, for any UI controls. It internally checks what language is currently active on the FlexPendant and uses the correct font for that language.

**NOTE!**

If you use other fonts than available on the FlexPendant, that is, `Tahoma` and `Courier New`, the application will use `Tahoma` instead of the intended one.

#### 5.3.6. The use of icons

---

##### Overview

It is easy to display icons, images and photos on the FlexPendant touch screen. Utilizing this possibility is recommended, as many people find it easier to read pictures than text. Intuitive pictures can also be widely understood, which is a real advantage in a multinational setting.



##### NOTE!

The operating system of the first generation FlexPendant device (SX TPU 1) does not support images of more than 256 colors.



##### CAUTION!

Be aware of the limited memory resources of the FlexPendant device. Do not use larger images than necessary. Also see [How large can a custom application be? on page 166](#)

---

##### FlexPendant controls with images

Several FlexPendant controls support images, which are imported as bmp, jpg, gif or ico files when an **Image**, **Icon** or **BackgroundImage** property is set. These are some controls to be used with pictures:

- MenuItem
- Button
- TpsCtrlTitlePanel
- TabPage
- ListViewItem
- TpsForm
- FPRapidData, FpToolCalibration, FpWorkObjectCalibration
- GTPUSaveFileDialog

---

##### PictureBox and ImageList

There are also some Windows controls that can be used to display images.

`Windows.Forms.PictureBox` is one such control. It can display graphics from a bitmap (.bmp), icon (.ico), JPEG or GIF file. You set the **Image** property to the preferred image either at design time or at run time.

`Windows.Forms.ImageList` can manage a collection of **Image** objects. The `ImageList` is typically used by another control, such as the `ListView` or the `TabControl`. You add bitmaps or icons to the `ImageList`, and the other control can choose an image from the `ImageList` index. How to use images for a `TabControl` is detailed in [How to add tab images on page 91](#).



*Continued*

---

#### The TpsIcon class

The `TpsIcon` class offers static references to a number of icons used in the standard FlexPendant applications. You may want to use some of these icons if they suit the application need. `TpsIcon` belongs to the namespace `ABB.Robotics.Tps.Drawing`.



#### **TIP!**

In the *Reference Manual FlexPendant SDK*, click the **Search** tab and search for *TpsIcon Members* to get a short description of the ABB icons that are available as static properties.

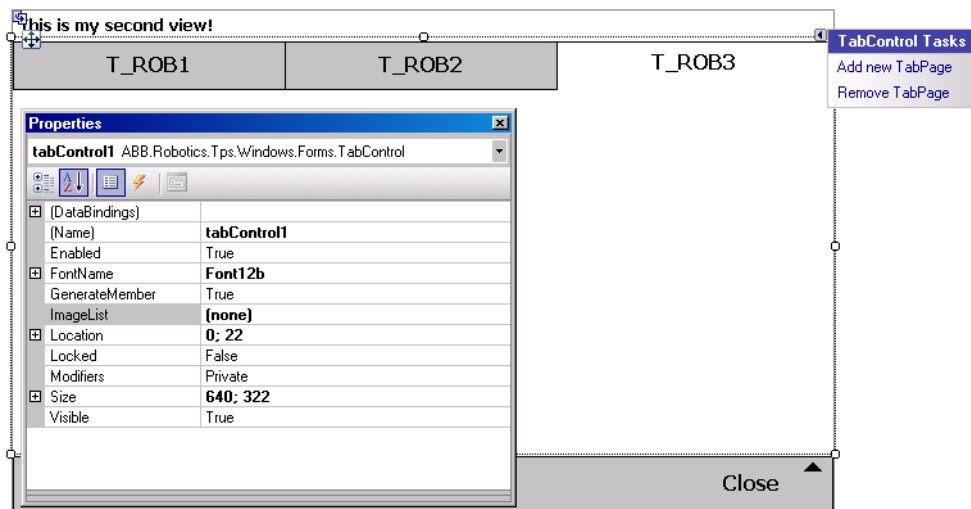
### 5.3.7. TabControl

#### Overview

The `ABB.Robotics.Tps.TabControl` is used and implemented much in the same way as a regular `System.Windows.Forms.TabControl`. The possibility to use tab icons, however, is an additional feature of the ABB `TabControl`.

#### Illustration

The following screenshot shows a `TabControl` and its **Properties** window. To add or remove tab pages you click the little arrow in the top right corner of the control or click **Add new TabPage** at the bottom of the **Properties** window.



6.2.4\_1

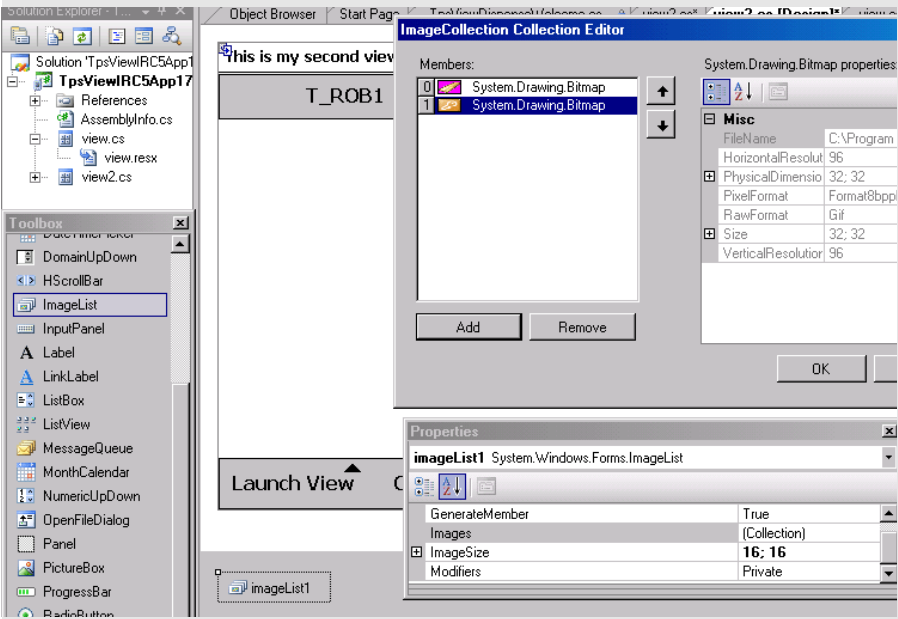


#### NOTE!

The ABB `TabControl` has no `TabPage` property. Depending on where in the `TabControl` you click, you select either the entire `TabControl` or one of the `TabPage`s. To view the **Properties** window for the `T_ROB1` `TabPage` in the figure, you would have to click the `T_ROB1` tab and then the `T_ROB1` page following the tab.

*Continued***How to add tab images**

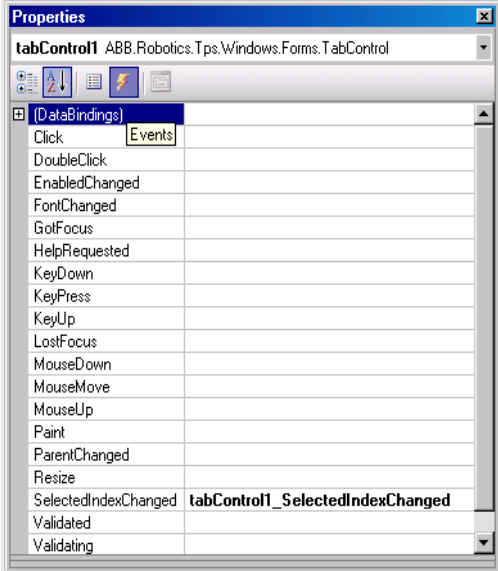
The following procedure explains how you use the **ImageList** property to add icons to the tabs of the TabControl:

Step	Action
1	Drag a <code>System.Windows.Forms.ImageList</code> to the <b>Designer</b> area. As you will notice it has no visual representation, but is placed on the components pane following the container control.
2	Display the <b>Properties</b> window of the ImageList.
3	Click the property <b>Images</b> and then the browse button, which will appear.
4	Click <b>Add</b> and import an image. Repeat until you have added the icons your TabControl needs.
	 <p>6.2.4_2</p>
5	Now display the <b>Properties</b> window of the TabControl and select your ImageList as the <b>ImageList</b> property.
6	Select one of the tab pages and set the property <b>ImageIndex</b> , which defines which of the images in the ImageList is to be used for the tab. Repeat this procedure for all tab pages.

Continued

#### How to add an event handler using the Properties window

It is easy to attach an event handler to a control by using the Properties window. This procedure shows how to make something happen when you click a tab to select another tab page:

Step	Action
1	In the <b>Designer</b> , select the <code>TabControl</code> and display the <b>Properties</b> window.
2	Display the events available for the control by clicking the <b>Events</b> button.  6.3.4_3
3	Select the <b>SelectedIndexChanged</b> event and double click. A method name is now auto generated for the <b>SelectedIndexChanged</b> event. The code view takes focus, the cursor placed inside the generated event handler.
4	Add code to make something happen. For now this code will do: <pre>this.Text = "TabControl notifies change of tab pages";</pre> This will change the title text of the <code>TpsForm</code> when a tab is clicked.
5	Test the functionality by building, deploying and starting the application. For more information, see <a href="#">step 5-8 in Hands on - Hello world on page 68</a> .



#### NOTE!

The subscription to the event has been done under the hood, and you do not need to bother about it. If you use `C#` you will notice that some code adding the subscription has been inserted in the `InitializeComponent` method:

```
this.tabControl1.SelectedIndexChanged += new  
System.EventHandler(this.tabControl1_SelectedIndexChanged);
```

#### Disposing TabControl

You do not need to explicitly call `Dispose` on the `TabControl` object. The reason is that `InitializeComponent` adds the tab pages to the controls collection of the `TabControl`, and the `TabControl` itself to the `control` collection of the container class.  

```
this.Controls.Add(this.tabControl1);
```

The `TabControl` is thus removed when the `Dispose` method of your container class calls `Dispose` on its base class like this: `base.Dispose(disposing);`

## 5.3.8. Button, TextBox and ComboBox

### Overview

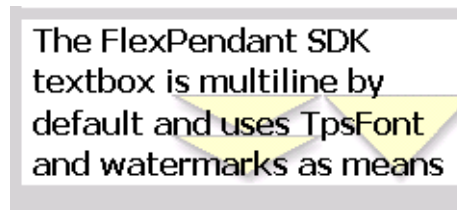
The FlexPendant button, text box and combo box controls are not very different from the equivalent Windows controls.

### Using Button

A common size of a FlexPendant `Button` is 120 \* 45 pixels; it can be a lot smaller than its default size. As opposed to a Windows button it can display an image. As pointed out in [Illustration on page 90](#) you use the property **BackgroundImage** and browse for the image to be displayed.

### Using TextBox

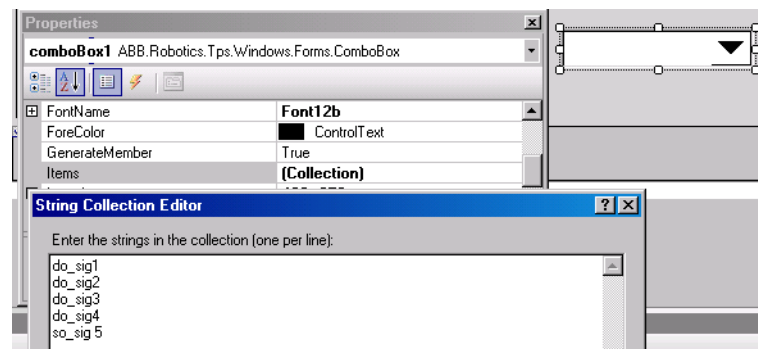
The `TextBox` control is multiline by default and uses the typical FlexPendant scroll bars as shown in the following figure.



6.2.7\_1

### Using ComboBox

The `ComboBox` is very similar to the `System.Windows.Forms.ComboBox`, but is visually improved to suit the FlexPendant touch screen. It can be statically populated using the **Items** property as illustrated by the figure. If you wish to populate it dynamically, you need to write code for it.



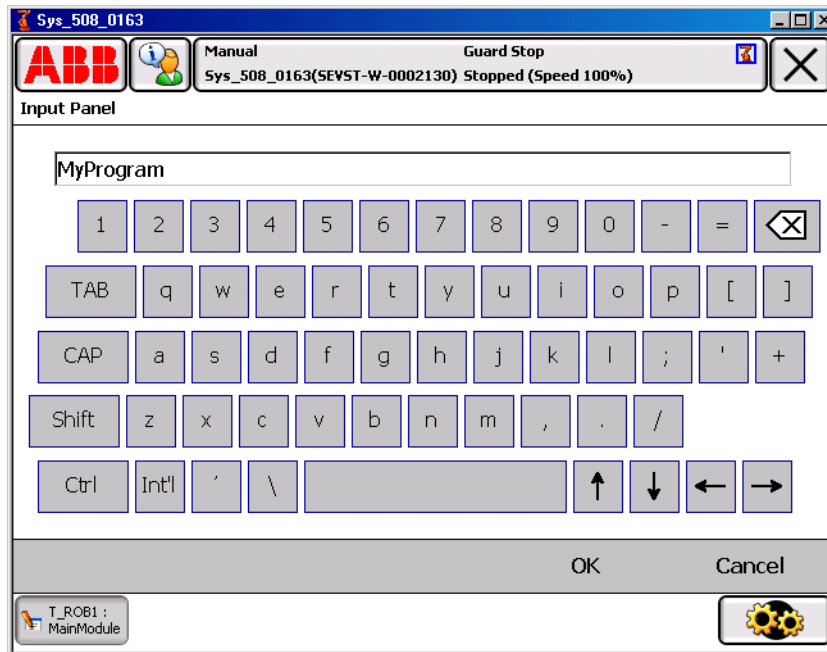
6.2.7\_2

`SelectedIndexChanged` is the most commonly used among the events. It occurs when you select another item from the list.

### 5.3.9. AlphaPad

#### Overview

The AlphaPad control is a virtual keyboard, which enables you to enter strings. It covers the whole FlexPendant display. The figure shows its appearance in run time.



6.2.10\_1

The AlphaPad has no graphical representation in design-time. When dragged from the **Toolbox** to the **Designer** it is placed in the components pane, as shown in the figure in the next section. Code is generated for it similar to controls that are visually laid out on the form in design-time.

#### Launching the AlphaPad

You need to write some code to launch the virtual keyboard. The following code explains how you do it:

C#:

```
this.alphaPad1.ShowMe(this);
```

VB:

```
Me.AlphaPad1.ShowMe(Me)
```



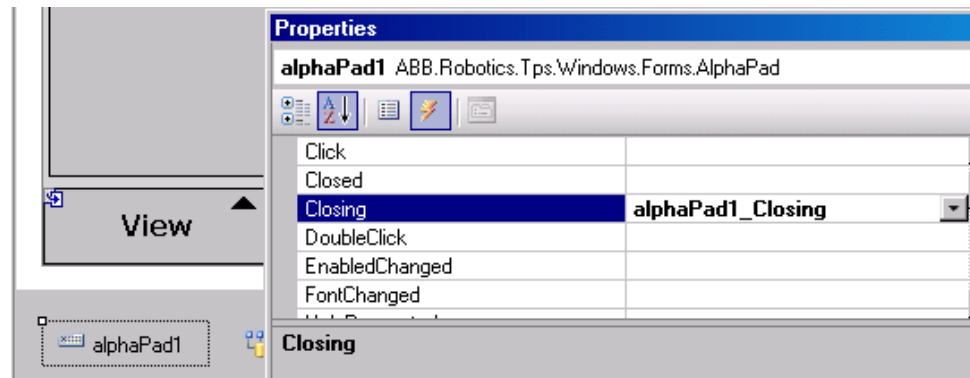
#### NOTE!

The code for launching the AlphaPad should be executed by an appropriate event handler.

### Adding event handlers

The **Properties** window can be used to program AlphaPad event handlers. You double click the event you want to respond to. This takes you to the code editor, the cursor placed inside the auto generated event handler, where you can write code to save user input to a member variable for example:

```
if (alphaPad1.SelectedText != string.Empty)
{
    _userInput = alphaPad1.SelectedText;
}
```



6.2.10\_2

### Validating the result at the Closing event

The `Closing` event is generated when the **OK** or **Cancel** button of the AlphaPad is pressed. You can use this event to validate the string entered by the user. You can set the **Cancel** property of the event argument to true if you want the AlphaPad to remain open until a valid input value has been entered:

VB:

```
Private Sub NamePad_Closing(sender As Object, e As
    System.ComponentModel.CancelEventArgs) Handles
    NamePad.Closing
    If NamePad.SelectedText.CompareTo("No Name") = 0 &&
        NamePad.DialogResult =
        System.Windows.Forms.DialogResult.OK Then
        e.Cancel = True
    End If
End Sub
```

C#:

```
private void namePad_Closing(object sender,
    System.ComponentModel.CancelEventArgs e)
{
    if ((this.namePad.SelectedText.CompareTo("No Name") == 0) &&
        (this.namePad.DialogResult ==
        System.Windows.Forms.DialogResult.OK))
    {
        e.Cancel = true;
    }
}
```

*Continues on next page*

*Continued*

---

#### Using the result at the Closed event

The Closed event has to be caught by an event handler, as the object cannot be disposed of until it has been closed. The result may be retrieved in this event handler or in the Closing event handler. First check that OK and not Cancel was pressed, then retrieve user input. Finally the AlphaPad should be disposed of.VB:

```
Private Sub NamePad_Closed(sender As Object, e As EventArgs)
    Handles NamePad.Closed
    If NamePad.DialogResult = Windows.Forms.DialogResult.OK Then
        Me.answer = NamePad.SelectedText
    End IfNamePad.Dispose()
End Sub
```

C#:

```
private void namePad_Closed(object sender, EventArgs e)
{
    if (this.namePad.DialogResult ==
        System.Windows.Forms.DialogResult.OK)
    {
        this.answer = this.namePad.SelectedText;
    }
    this.alphaPad1.Dispose();
}
```

---

#### Removing the AlphaPad control

The AlphaPad is NOT added to the control collection, and will therefore NOT be disposed of by the base class of its container.

You are responsible for explicitly calling its `Dispose` method when it is no longer used. In the preceding example, this is done at the `Closed` event. This implies that a new AlphaPad instance is created the next time its launch event is triggered.

Another way of dealing with this is to let the instance created by the Designer live until its container view is closed. This alternative means destroying it in the `Dispose` method of the container class:

```
this.alphaPad1.Dispose();
```



#### CAUTION!

If you forget to call `Dispose` on an AlphaPad control you will have a memory leak. For further information see [GUI controls and memory management on page 76](#).



## 5.3.10. ListView

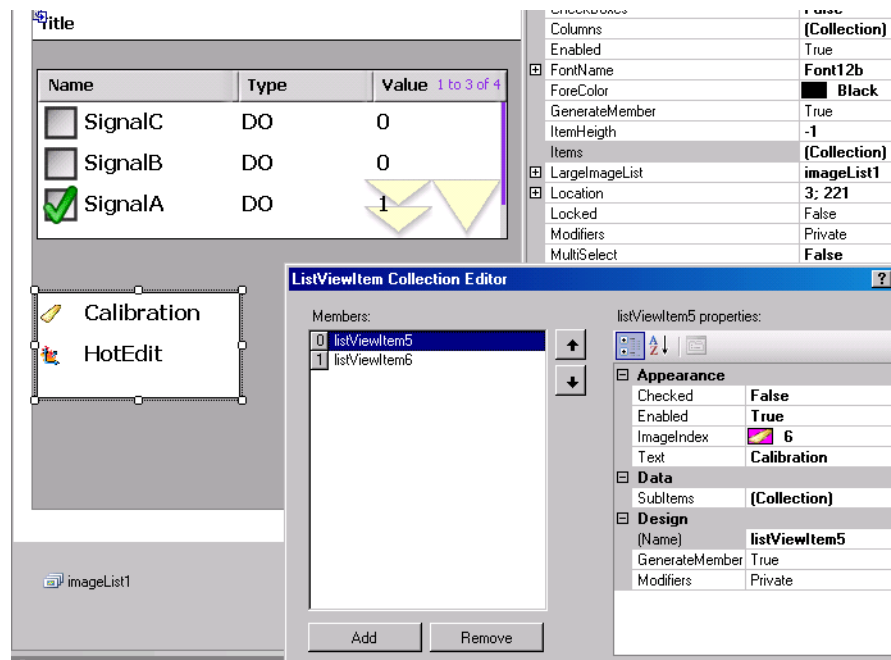
### Overview

The `ABB.Robotics.Tps.Windows.Forms.ListView` is very similar to a standard `.Net ListView`. The main difference is that its appearance is somewhat adapted to the use of a touch screen, as can be seen in the figure.

You can use the `ListView` control in a variety of ways. Usually you use it to display a list of items with item text and, optionally, an icon to identify the type of item. By using the `CheckBoxes` property it is also possible to have a check box appear next to each item in the control, allowing users to check the items they want to perform an action on.

### Illustration

The figure shows the Designer. You see two different `ABB ListView` lists, a standard `.Net ImageList` in the components pane under the lists and the **Properties** window for the *Calibration, HotEdit* list.



6.3.10\_1

*Continued*

---

#### Using properties to control appearance

The first list has **CheckBoxes** set to true. To have columns displayed **View** must be set to **Details**. The columns are created by selecting **Columns** and adding any number of columns in the **ColumnHeader Collection Editor**. **Scrollable** has been set to true to enable you to scroll the list to see all items. To have the current number of list items displayed **ShowNumberOfItems** has been set to true. By using **Sorting** the list items can be alphabetically sorted, in this case in **Descending** order. The list is statically populated, using the **Items** property.

The **View** property of the other list is set to **LargeIcon** instead of **Details**. To display icons a standard .Net **ImageList** is used. You first launch the **Properties** window for the **ImageList** and add the images you want to use. At the **LargeImageList** property of the **ListView** you select the image list, and for each **Item** you select the **ImageIndex** to use, as is shown in the **ListViewItem Collection Editor**.

Using the **Properties** window you can also add event handling. The most commonly used event is **SelectedIndexChanged**, which is triggered when a new list item is selected.

The **Properties** window can be used to statically populate the list. Usually, however, lists are populated dynamically. In *Getting signals using SignalFilter on page 151* there is a code example, which shows how I/O signals are dynamically displayed in an ABB **ListView**.

---

#### ABB specific properties

These properties are specific for the ABB **ListView**:

Property	Usage
MultiSelect	Specifies whether multiple items in the control can be selected. The default is false.
Scrollable	Specifies whether a scroll bar is added to the control when there is not enough room to display all items. The default is true. <b>Note!</b> The typical FlexPendant scroll bars are used by the first list in the preceding figure.
SelectionEnabledOver-Scrollbuttons	Specifies whether a touch inside the scroll region should select an item or scroll the list. The default is false.
ShowNumberOfItems	Specifies whether the current number of items in the list is displayed. The default is true. Scroll bars should be added to the control to allow you to see all the items. If the list is not scrollable the default value is false.
ShowSelection	Specifies whether selection is shown or not. The default is false.

### 5.3.11. CompactAlphaPad and NumPad

#### Using CompactAlphaPad

The CompactAlphaPad control is used to enable user input. The input can be in the form of capital letters or numbers depending on the property `ActiveCharPanel`, which can be set to `Characters` or `Numeric`. It has a fixed size, big enough for the use of a finger to click its keys. There are properties available to define whether the numerical panel should support numerical operands, special characters, space and comma. All these properties are set to true in the figure:



6.3.11\_1

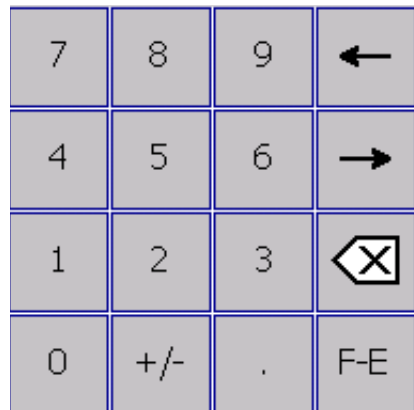
In order for your application to use the user input, you must connect to a text box, which displays the text or figures entered. This connection has to be coded manually, for example in the constructor, after the call to `InitializeComponent`, for example:

```
this.compactAlphaPad1.Target = this.textBox1;
```

#### Using NumPad

The NumPad is very similar to the CompactAlphaPad. You have to create a `Target` to use the input, for example:

```
this.numPad1.Target = this.textBox2;
```



6.3.11\_2

### 5.3.12. GTPUMessageBox

---

#### Overview

`GTPUMessageBox` is the message box control to be used by FlexPendant applications. You should not use the standard `.NET MessageBox`.

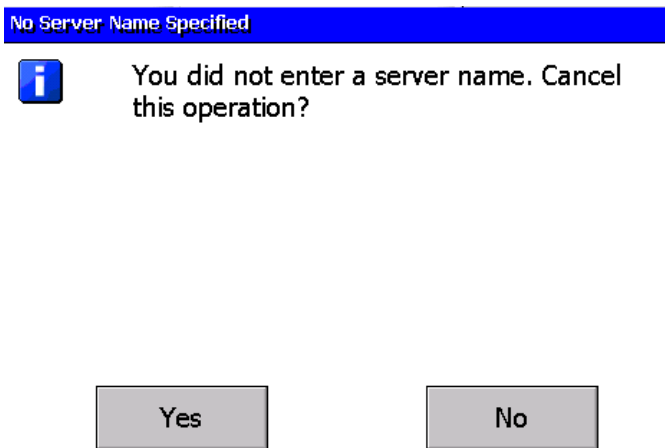
#### Design issues

The `GTPUMessageBox.Show` arguments include an owner argument, which is the control that displays the `MessageBox`, and a callback, which is a `MessageBoxEventHandler` called when the `MessageBox` is closed.

Except this, it is used in the same way as the regular Windows `MessageBox`. It is used together with the regular Windows `MessageBoxButtons` and `MessageBoxIcon`.

#### Simple code example

The following screenshot and code show how to display a simple message on the FlexPendant using the `GTPUMessageBox`.



6.2.6\_1

```
string message = "You did not enter a server name. Cancel this  
operation?";  
string caption = "No Server Name Specified";  
GTPUMessageBox.Show(this, null, message, caption,  
    System.Windows.Forms.MessageBoxIcon.Question,  
    System.Windows.Forms.MessageBoxButtons.YesNo);
```



#### NOTE!

Sometimes it can be a bit tricky to get the first argument right. The owner might be `this`, `this.Parent` or even `this.Parent.Parent`.

---

## Using a callback

The logics in your program determines if there is any need for a callback. In the previous example the callback argument was `null`. The message box will close down after the user has answered the question, no matter if the answer is **Yes** or **No**.

If we think about the message, it seems likely however, that something should happen if you click **No**. Let us change the implementation and use a callback for this purpose:

```
GTPUMessageBox.Show(this, new
    MessageBoxEventHandler(OnServerMessageClosed), message,
    caption, System.Windows.Forms.MessageBoxIcon.Question,
    System.Windows.Forms.MessageBoxButtons.YesNo);

//implement callback
private void OnServerMessageClosed(object sender,
    ABB.Robotics.Tps.Windows.Forms.MessageBoxEventArgs e)
{
    if(e.DialogResult == System.Windows.Forms.DialogResult.No)
    {
        //Use default server...
    }
}
```

If we really do not want to take action depending on how the user responds, it makes more sense to use **OK** instead of **Yes** and **No** as message box buttons, e.g:

```
GTPUMessageBox.Show(this, null, "You are not allowed to perform this
operation, talk to your system administrator if you need access",
"User Authorization System", MessageBoxIcon.Hand,
MessageBoxButtons.OK);
```



### NOTE!

The last example uses a better message box title then the previous ones. The title preferably should tell you from which part of the system the message originates.

### 5.3.13. GTPUFileDialog

---

#### Overview

The FlexPendant SDK provides a number of file dialog box used by the end-user to interact with the file system of the robot controller. These controls all inherit the abstract class `GTPUFileDialog` and belong to the namespace `ABB.Robotics.Tps.Windows.Forms`.

#### File dialog types

There are three different types of file dialog controls:

Use...	when you want to...
<code>GTPUOpenFileDialog</code>	Enables you to specify a file to be retrieved from the controller file system.
<code>GTPUSaveFileDialog</code>	Enables you to specify a file to be saved to the controller file system.
<code>GTPUFolderBrowserDialog</code>	Enables you to specify a folder on the controller file system.



#### NOTE!

The Open/Save/Browse file dialog box represent a convenient way for you to specify folder and filename for a file operation. You should know, however, that the dialog box themselves do not perform any file operation, they only provide the controller file system path to be used by your application.



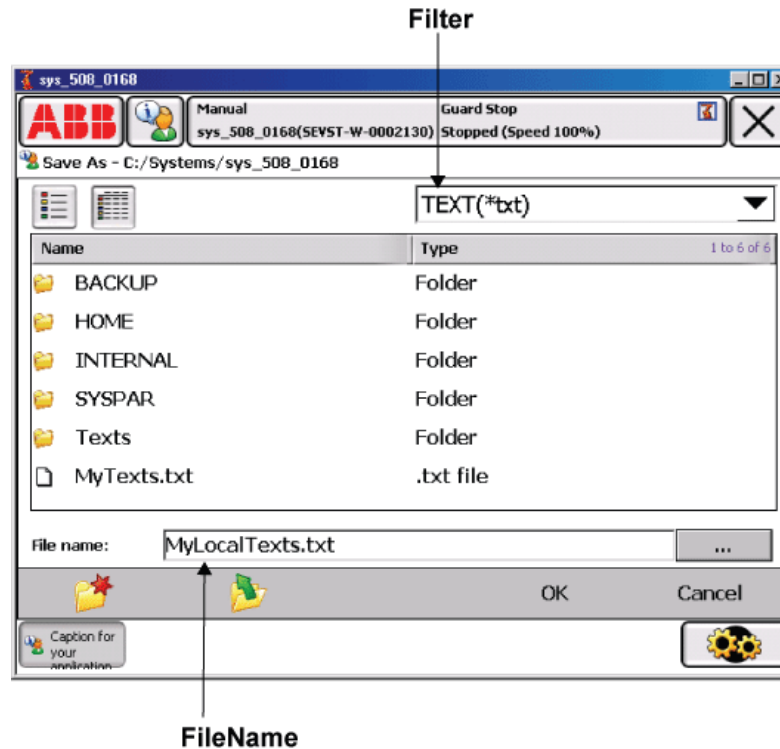
#### CAUTION!

When added to the Visual Studio Designer from the Toolbox, these controls will be placed in the components pane. They must be explicitly removed by a call to their `Dispose` method, for example in the `Dispose` method of the class that created them.

Continued

Illustration

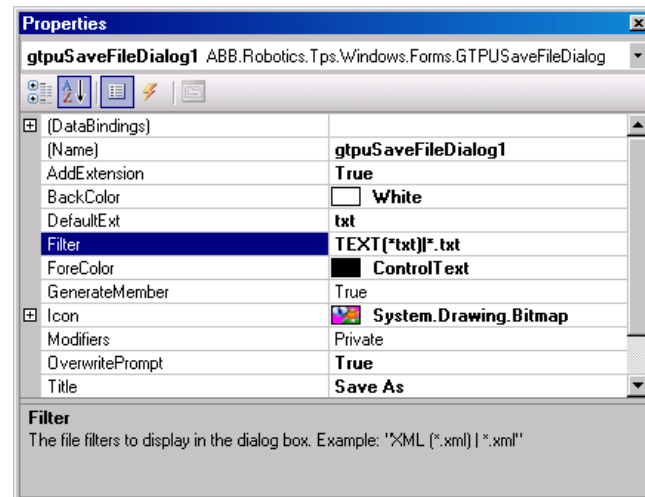
The following is a screenshot of the `GTPUSaveFileDialog`. The other file dialog box have almost the same appearance and the way they work is very similar. Using the icons of the command bar you create new folders and browse the controller file system. Together with the list and the textbox, you specify a remote path. The **FileName** property should be read to retrieve this path. It returns the complete path to the controller file system, even though only the file name is visible in the **File name** textbox



6.3.13\_1

**Note!** The button after the **File name** textbox opens the virtual keyboard, enabling you to change the name of the file to be saved. The **Filter** property of the control is set to `TEXT(*.txt)|*.txt`. The first part of the setting is displayed in the filter textbox.

This is the Properties window of this dialog:



6.3.13\_2

Continues on next page

Continued

#### Implementation details

The **Properties** window gives a clear description of a selected property, as can be seen in the preceding figure. Here only a few important properties will be detailed:

Property	Details
Filter	Carefully observe the string format when you set this property, for example: Program Files (*.pgf)*.pgf The first part is displayed in the combo box and the second part is used by the SDK to retrieve the correct files to be displayed in the list. You can also specify several filters, which the user can choose from. Program Files (*.pgf)*.pgf All Files (*.*)*.*;
FileName	Cannot be accessed in design-time by using the <b>Properties</b> window, but should be manually coded when the file dialog is launched if it is a save file dialog. When the dialog is closed, you should read this property. It holds the remote path and file name of the file to be opened, or the remote path and file name of the file to be saved. <b>Note!</b> Remember that <i>remote</i> refers to the controller file system and <i>local</i> to the FlexPendant file system.
InitialDirectory	Cannot be accessed in design-time by using the <b>Properties</b> window. Specifies the initial directory to be displayed by the file dialog box.



#### NOTE!

For your program to use the controller path selected by the end user, you need to read the **FileName** property at the `Closing/Closed` event of the file dialog.

#### Example

This piece of code sets `InitialDirectory` and `FileName`, then sets up a subscription to the `Closed` event and finally displays the `GTPUSaveFileDialog`

```
saveFileDialog.InitialDirectory = initialDir;  
saveFileDialog.FileName = programName;  
saveFileDialog.Closed += new  
    EventHandler(SaveProgram_FileDialog_EventHandler);  
saveFileDialog.ShowMe(_parent);
```

The code of the `SaveProgram_FileDialog_EventHandler` method retrieves the specified path of the remote file system, including the file name, by reading the `FileName` property:

```
string remotePath = saveFileDialog.FileName;
```

#### NOTE!

The file has not yet been saved to the robot controller. To do that you should call `FileSystem.PutFile` using the retrieved path as the *remoteFile* argument. Likewise, to load a specified remote file to the FlexPendant file system you should use the retrieved path in the call to `FileSystem.GetFile`.





### 5.3.14. Data of RAPID data and IO signals

#### What is databinding?

Databinding is the process of binding a property of a GUI control to a data source, so that the property automatically reflects the value of the data source. In .NET CF 2.0 this functionality was simplified with the `BindingSource` class. This class encapsulates the complexity related to setting up and managing databinding. It allows you to connect a `RapidDataBindingSource` or a `SignalBindingSource` to a GUI control without having to write a single line of code. (Except the `Dispose` call when the binding sources are no longer needed.)

#### FlexPendant SDK classes to be used as binding sources

In the FlexPendant SDK there are two classes that inherit .NET `BindingSource`: `RapidDataBindingSource` and `SignalBindingSource`. These classes enable binding to RAPID data and IO signals respectively. They belong to the `ABB.Robotics.DataBinding` namespace and the assembly you need to reference is `ABB.Robotics.DataBinding.dll`.

#### RapidDataBindingSource

By using `RapidDataBindingSource` an automatic update of the bound GUI control takes place when the value of the specified RAPID data source in the controller has changed, or when the control is repainted.

#### SignalBindingSource

By using `SignalBindingSource` an automatic update of the bound GUI control takes place when the value of the specified IO signal in the controller has changed, or when the control is repainted.



#### NOTE!

It is only possible to use persistent RAPID data (PERS) as data source.



#### NOTE!

If you want to let users modify RAPID data, launching the standard FlexPendant application *Program Data* from your application is probably the best alternative. For more information, see *Using standard dialog box to modify data on page 114*.

#### GUI example

Many different controls can be used with binding sources, for example `TpsLabel`, `TextBox`, `ListView` and so on. The following screenshot shows two `System.Windows.Forms.DataGrid` controls that each bind to several objects defined in a `RapidDataBindingSource` and a `SignalBindingSource`. The labels in each `GroupBox` are bound to the same sources. As you see the grid displays all objects defined in the `BindingSource` control, whereas each label displays the currently selected grid row.

When a signal or a data value changes this GUI is automatically updated with the new value. You do not have to write any code make this happen, as setting up subscriptions and updating the user interface is done by the underlying implementation.

*Continues on next page*

## 5 Using the FlexPendant SDK

### 5.3.14. Data of RAPID data and IO signals

Continued

The screenshot shows the 'Virtual FlexPendant' application window. The title bar includes the ABB logo, a manual icon, and the text 'Manual Guard Stop sys\_508\_0168(SEV5T-W-0002130) Stopped (Speed 100%)'. Below the title bar, the main area is titled 'FP SDK BindingSource'. It is divided into two sections: 'My RAPID data' and 'My IO signals'. The 'My RAPID data' section contains a table with columns 'Value', 'Variable', 'Module', and 'Task'. The third row is selected, showing '3', 'Num3', 'user', and 'T\_Rob1'. To the right of this table are input fields for 'Current value: 3', 'Variable name: Num3', 'Module name: user', and 'Task name: T\_Rob1'. The 'My IO signals' section contains a table with columns 'Type', 'Name', and 'Value'. The second row is selected, showing 'DigitalO', 'MyD02', and '0'. To the right are input fields for 'Signal type: Digital...', 'Signal name: MyD02', and 'Current value: 0'. At the bottom left, there is a placeholder for a caption, and at the bottom right, there is a gear icon.

Value	Variable	Module	Task
122	myNum	user	T_RO
1	Num2	user	T_Rol
3	Num3	user	T_Ro

Type	Name	Value
DigitalO	MyD01	1
DigitalO	MyD02	0
DigitalO	MyD03	0

6.3.14\_1



#### CAUTION!

To avoid any memory leaks an explicit call to the `Dispose` method of `BindingSource` controls must be made. However, the wrapper-objects `SignalObject` and `RapidDataObject` created for you are disposed of under the hood, so you do not need to worry about it.

**How to use the Visual Studio designer for**

This section explains how to create the FlexPendant view shown in the previous section. First a `RapidDataBindingSource` control with bindings to specified RAPID data is created. Then the **DataBindings** property of a `TpsLabel1` is used to bind the label to the binding source. Finally a standard .NET `DataGrid` is bound to the same binding source.

Step	Action
1.	Start by dragging a <code>RapidDataBindingSource</code> from the Toolbox to the Designer. It will be placed in the Components pane under the form. Open the <b>Properties</b> window and select the <b>RapidDataList</b> property to add the RAPID data you are interested in. For each new <b>RapidDataObject</b> member you must specify module name, task name and name of the persistent RAPID data to bind.

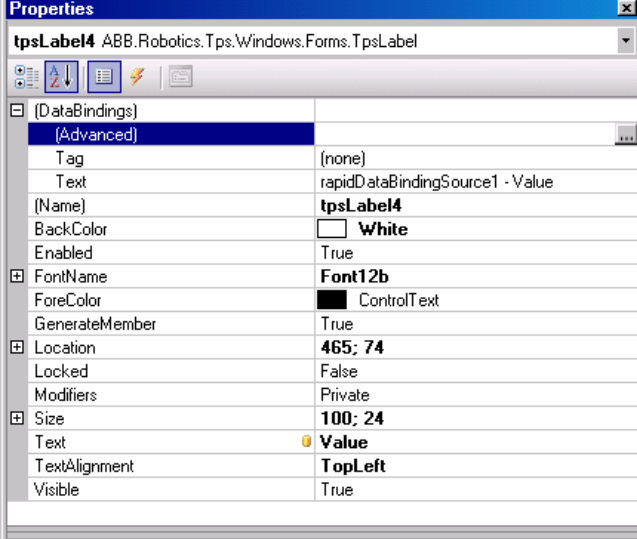
The screenshot shows the Visual Studio designer interface. At the top, there is a 'My RAPID data' table with columns 'Value', 'VariableName', 'ModuleName', and 'TaskName'. Below it is a 'My IO signals' table with columns 'Type' and 'Name'. To the right, there are input fields for 'Current value:' (containing 'Value') and 'Variable name:' (containing 'Name'). In the center, the 'Properties' window for 'rapidDataBindingSource1' is open, showing the 'RapidDataList' property set to '(Collection)'. At the bottom, the 'RapidDataObject Collection Editor' window is open, showing a list of four 'RapidDataObject' members. The first member is selected, and its properties are shown on the right: 'ModuleName' is 'user', 'TaskName' is 'T\_ROB1', and 'VariableName' is 'myNum'. The bottom left corner of the designer shows the text '6.3.14\_2'.

## 5 Using the FlexPendant SDK

### 5.3.14. Data of RAPID data and IO signals

*Continued*

Step	Action
2.	The next step is to open the <b>Properties</b> window for the label that is to display the value of the RAPID data. Expand the <b>DataBindings</b> node and select <b>Advanced</b> .



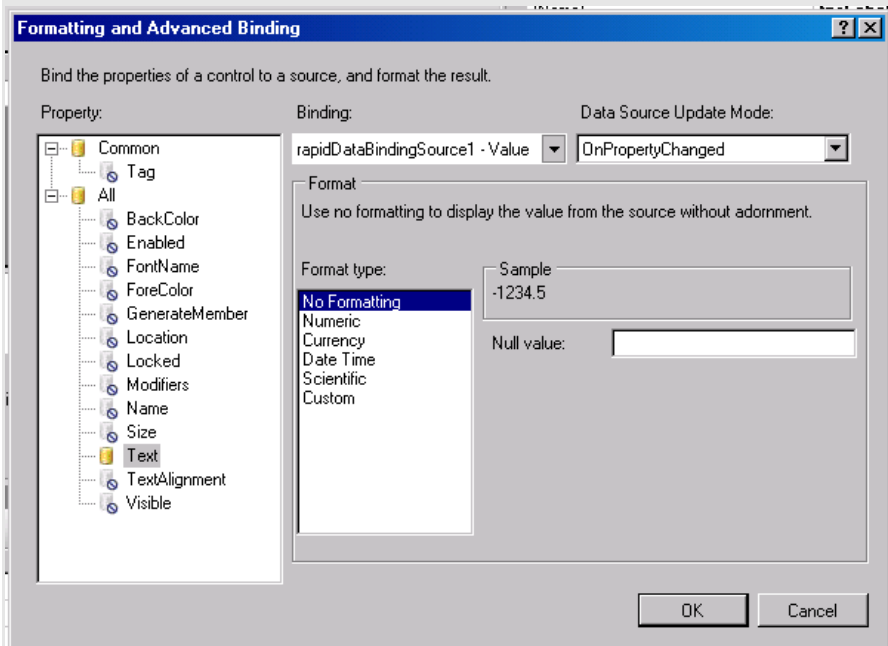
The screenshot shows the Properties window for a control named **tpsLabel4**. The **DataBindings** node is expanded, and the **(Advanced)** tab is selected. The **Text** property is set to **rapidDataBindingSource1 - Value**. Other visible properties include **Tag** (none), **(Name)** (tpsLabel4), **BackColor** (White), **Enabled** (True), **FontName** (Font12b), **ForeColor** (ControlText), **GenerateMember** (True), **Location** (465; 74), **Locked** (False), **Modifiers** (Private), **Size** (100; 24), **Text** (Value), **TextAlignment** (TopLeft), and **Visible** (True).

6.3.14\_3

Continued

Step	Action
------	--------

- |    |  |
|----|--|
| 3. | <p>In the <b>Advanced Binding</b> dialog box that appears, choose the already created <code>RapidDataBindingSource</code> in the <b>Binding</b> combo box, at the same time specifying which one of the <b>RapidDataObject</b> properties you want to bind to, in this case <b>Value</b>. (The other properties available are variable name, task name and module name, as can be seen in the figure in step 1.)</p> <p>You also select your preferred <b>Data Source Update Mode</b>, usually <i>OnProperty-Changed</i>. The yellow marking in the list to the left shows that the binding has been done to the <b>Text</b> property of the label. When a control has been bound to a data source you will see the same yellow marking in its <b>Properties</b> window, at the bound property. For more information, see <i>illustration on step 2</i>.</p> |
|----|--|



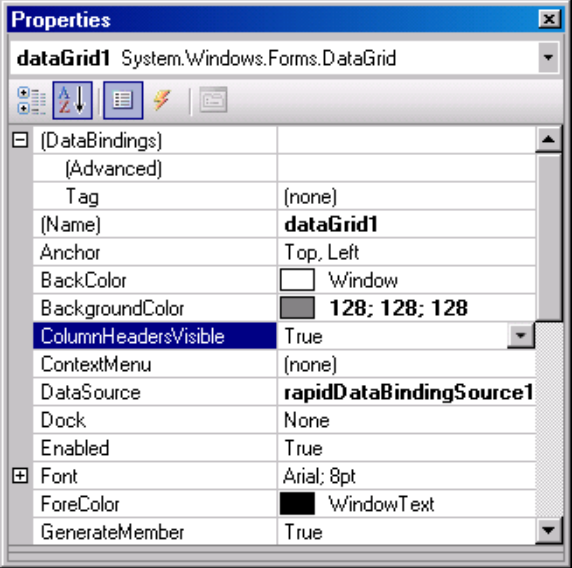

6.3.14\_4



**NOTE!**

If a label has been bound to a data source with several member objects, the first one in the list is the one by default displayed by the label. If the selected index of the list changes (if you select a row in the grid for example) the label is updated automatically to show the value of the selected index. For more information, see *the illustration in GUI example on page 105*.

Continued

Step	Action
4.	<p>Now launch the <b>Properties</b> window of the <code>DataGrid</code> control. Set <b>ColumnHeaderVisible</b> to true and select your data source at the <b>DataSource</b> property.</p>  <p>6.3.14_5</p> <p> <b>NOTE!</b></p> <p>The <code>DataGrid</code> control displays only one row in design-time (For more information, see <i>the illustration of step 1</i>). In run-time, however, the entire collection of <b>Rapid-DataObject</b> members is displayed (see the figure in <a href="#">GUI example on page 105</a>).</p>

### SuspendBinding/ResumeBinding

`SuspendBinding()` and `ResumeBinding()` have extended functionality compared to the methods of the inherited `BindingSource` class. Removing/adding subscriptions have been added, as subscribing to signals and RAPID data can be resource consuming. These methods can be used in the `Deactivate/Activate` methods, which are executed each time you switch back and forth between different applications using the FlexPendant task bar. For more information, see [Application Framework usage - ITpsViewActivation on page 171](#).

If you suspend the binding you no longer get any updates if the data source is changed. However, the binding still exists, so if **DataSourceUpdateMode** is set to `OnValidation` the value will be updated when you repaint the control, as the value from the controller will be read before it is repainted.

Method	Description
<code>SuspendBinding</code>	Suspends the binding and value change event.
<code>ResumeBinding</code>	Resumes the binding and value change event.

## 5.4 Launching other views

### 5.4.1. Using launch service

#### Overview

The FlexPendant SDK provides a launch service, which enables a FlexPendant SDK application to start a standard ABB application, such as the program editor or the jogging view.

It is also possible to start another custom application by specifying proxy assembly name and the application view class.

For more information about how to add and launch another view of your own application, see [Adding a view to a custom application on page 83](#).



#### NOTE!

To launch a view in order to edit a specified rapid data instance, another mechanism is available. For more information, see [Using standard dialog box to modify data on page 114](#).

#### Installing ITpsViewSetup

It is only classes that inherit the `ITpsViewSetup` interface that can launch other applications. The `Install` method, which is called when your custom application is launched, has a sender argument. Your application needs to save this object to use the `ITpsViewLaunchServices` interface. It is used by the launch service to call `LaunchView` and `CloseView`.

```

VB:
//declaration
Private iTpsSite As ITpsViewLaunchServices
.....
//Install method of the TpsView class
Function Install(ByVal sender As System.Object, ByVal data As
    System.Object) As Boolean Implements ITpsViewSetup.Install
    If TypeOf sender Is ITpsViewLaunchServices Then
        // Save the sender object for later use
        Me.iTpsSite = DirectCast(sender, ITpsViewLaunchServices)
        Return True
    End If
    Return False
End Function

C#:
//declaration
private ITpsViewLaunchServices iTpsSite;
.....
//Install method of the TpsView class
bool ITpsViewSetup.Install(object sender,object data)

```

*Continues on next page*

## 5 Using the FlexPendant SDK

---

### 5.4.1. Using launch service

*Continued*

```
{
    if (sender is ITpsViewLaunchServices) {
        // Save the sender object for later use
        this.iTpsSite = sender as ITpsViewLaunchServices;
        return true;
    }
    return false;
}
```

---

#### Launching standard views

Using the `FpStandardView` enumerator, the following standard views can be started using the launch services:

- Program editor
- Program data
- Jogging
- Logoff
- Backup/Restore



#### TIP!

Launching the Program data view is the best way to let end-users create new RAPID data.



#### NOTE!

To launch the program editor an initialization object of `RapidEditorInitData` type can be used as argument. It specifies the task, module and row that the Program editor should display when it opens. For the other standard views no `InitData` can be used.

LaunchView / CloseView example

In the following example, the program editor is launched at a specified routine. First `initData` is created. Then the reference to the sender object, retrieved by the `Install` method in the previous example, is used to call `LaunchView`.

The `cookie` out argument is later used to specify the view to be closed by the `CloseView` method.

VB:

```
Dim initData As RapidEditorInitData = New RapidEditorInitData
    (ATask, AModule, ARoutine.TextRange.Begin.Row)
If Not (Me.iTpsSite.LaunchView(FpStandardView.RapidEditor,
    initData, True, Me.cookieRapidEd) = True) Then
    GTPUMessageBox.Show(Me, Nothing, "Could not start RapidEditor
    application")
    Return
End If
.....
Me.iTpsSite.CloseView(Me.cookieRapidEd)
```

© Copyright 2010 ABB. All rights reserved.

*Continues on next page*



*Continued*

C#:

```

RapidEditorInitData initData = new RapidEditorInitData(task,
    module, routine.TextRange.Begin.Row;
if (this.iTpsSite.LaunchView(FpStandardView.RapidEditor, initData,
    true, out this.cookieRapidEd) != true) {
    GTPUMessageBox.Show(this, null, "Could not start RapidEditor
        application");
    return;
}
.....
this.iTpsSite.CloseView(this.cookieRapidEd);

```

What happens if the specified view is already open when the call is made? The third argument specifies application behavior in this case. If `true`, another instance of the view is launched. If `false`, the already opened view gets focus. Notice, however, that creating a new instance is not possible for all standard views. The Jogging view, for example, is not allowed multiple instances.

---

### Launching custom applications

The launch service can also be used to launch another FlexPendant SDK application. The name of the proxy assembly along with the fully qualified class name should be used as arguments.

VB:

```

If Not (Me.iTpsSite.LaunchView("TpsViewCustomApp.gtpu.dll",
    "ABB.Robotics.SDK.Views.TpsViewCustomApp", Nothing, True,
    Me.cookieUser) = True) Then
    GTPUMessageBox.Show(Me, Nothing, "Could not start User
        application")
Return
End If

```

C#:

```

if (this.iTpsSite.LaunchView("TpsViewCustomApp.gtpu.dll",
    "ABB.Robotics.SDK.Views.TpsViewCustomApp", null, true, out
    this.cookieUser) != true) {
    GTPUMessageBox.Show(this, null, "Could not start User
        application");
    return;
}

```

**NOTE!**

It is the namespace of the *proxy* assembly, `ABB.Robotics.SDK.Views`, which should be used to specify the `TpsView` class.

**NOTE!**

By using the argument `cookieUser` your application can close the launched custom application in the same way as in the preceding *Launching standard views on page 112* example.



#### 5.4.2. Using standard dialog box to modify data

##### Overview

The `FpRapidData`, `FpToolCalibration` and `FpWorkObjectCalibration` dialog box take a `RapidData` reference as argument. When the dialog is opened it allows you to directly operate on the referenced `RapidData` object, for example modifying a RAPID data variable or calibrate a work object. These dialog box are the ones used by the standard ABB applications. They are ready-made and cannot be modified, except for the title.

##### Creating the dialog box

As with all secondary dialog box, the reference to the dialog should be declared as a class variable. This is to make sure that the reference is available for `Dispose`. The `RapidData` reference can be declared locally if it is disposed immediately after use, or else as a class variable. When the dialog has been created the title can be set using the `Text` property. A `Closed` event handler should be added to dispose the dialog. All three dialog box are created in the same way:

VB:

```
Private ARapidData As RapidData
Friend WithEvents FpRD As FpRapidData
.....
Me.ARapidData = Me.AController.Rapid.GetRapidData(ATaskName,
    AModuleName, AVariableName)
Me.FpRD = New FpRapidData(Me.ARapidData)
Me.FpRD.Text = Me.ARapidData.Name
AddHandler Me.FpRD.Closed, AddressOf Me.FpRD_Closed
Me.FpRD.ShowMe(Me)
```

C#:

```
private FpRapidData fpRD;
private RapidData aRapidData;
.....
this.aRapidData = this.aController.Rapid.GetRapidData(taskName,
    moduleName, variableName);
this.fpRD = new FpRapidData(this.aRapidData);
this.fpRD.Text = this.aRapidData.Name;
this.fpRD.Closed += new EventHandler( fpRD_Closed);
this.fpRD.ShowMe(this);
```

##### NOTE!

Verify that the `RapidData` is of correct `RapidDataType` for the dialog before using it, that is, `tooldata` for the `FpToolCalibration` dialog box and `wobjdata` for the `FpWorkObjectCalibration` dialog box. For more information, see [Type checking on page 115](#).



##### NOTE!

The `FpRapidData` dialog box is created with a `RapidData` as in argument. If the `RapidData` is an array, an index argument is used to specify which element should be displayed (the first element is 1).



#### Type checking

When calling the `FpToolCalibration` or `FpWorkObjectCalibration` constructor the `RapidData` value should be type checked before use:

VB:

```
If TypeOf Me.ARapidData.Value Is ToolData Then
    Me.FpTC = New FpToolCalibration(Me.ARapidData)
    .....
End If
```

C#:

```
if (this.aRapidData.Value is ToolData)
{
    this.fpTC = new FpToolCalibration(this.aRapidData);
    .....
}
```

## 5 Using the FlexPendant SDK

### 5.5.1. ABB.Robotics.Controllers

## 5.5 Using the Controller API

### 5.5.1. ABB.Robotics.Controllers

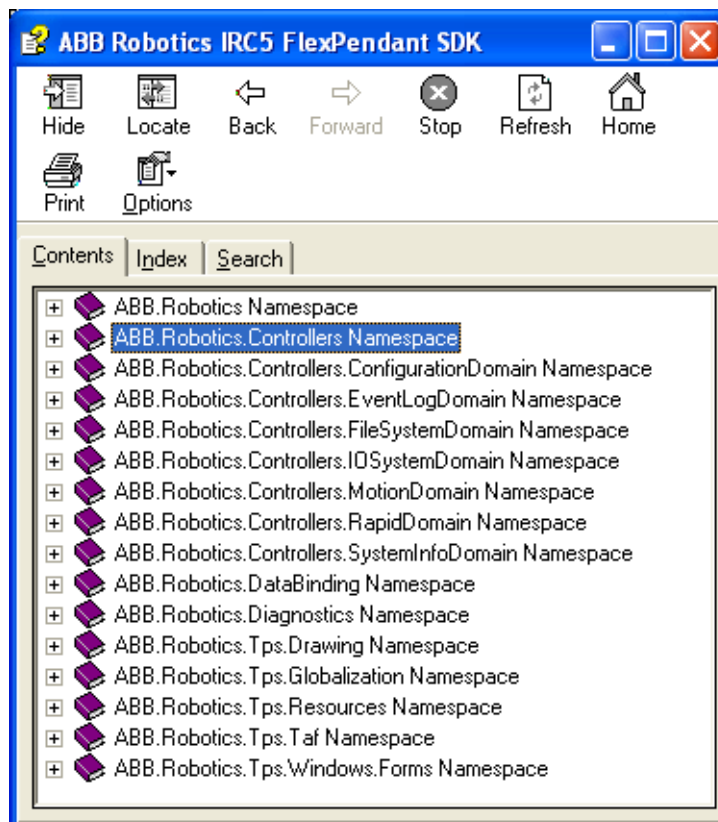
#### Controller API

To access the functions of the robot controller you utilize the class libraries of the FlexPendant SDK called *Controller Application Programming Interface* or *CAPI*. The assembly you need to reference to use controller functionality is *ABB.Robotics.Controllers.dll*.

#### CAPI domains

The top CAPI object is the `ABB.Robotics.Controllers.Controller`, which has to be created before any access to the robot controller.

The class libraries are organized in different domains (namespaces), as shown in the following screenshot, by the contents tab of the *Reference Manual FlexPendant SDK*. The name of a domain tells you something about the kind of services you can expect from it.



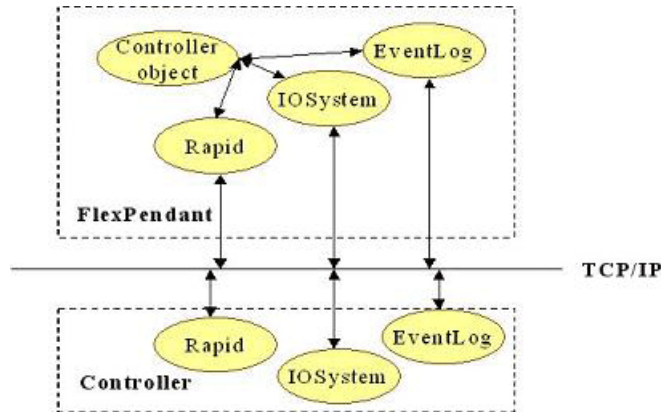
6.4.1\_1

© Copyright 2010 ABB. All rights reserved.

*Continues on next page*

**CAPI and controller domains**

This is a simplified illustration of how some ABB.Robotics.Controllers domains communicate with their respective domains in the robot controller:



6.4.1\_2



**NOTE!**

In the normal case, a single Controller object is created and used throughout the application, its lifetime corresponding to that of the application.

**What controller functionality is provided?**

Plenty of robot controller functionality is offered by the classes and methods of the FlexPendant SDK.

This table presents short descriptions of the kind of services that the different CAPI domains provide:

CAPI domain	Services
Controllers	Information about the controller, such as IP address, current user, Mac address, operating mode, controller state and so on. Notification when operating mode, state or mastership has changed. Backup and restore. Check if you have the required UAS grant and so on.
ConfigurationDomain	Read or write the value of a configuration parameter to the configuration database of the controller.
EventLogDomain	Notification when a new event log message has been written to the controller. Title, message, category, sequence number and time stamp of the message.
FileSystemDomain	Create, rename or remove files and directories in the controller file system. Retrieve a file from the controller and store it on the FlexPendant and vice versa.
IOSystemDomain	Read and modify I/O signals. Notify when a signal value has changed.
MotionDomain	Get/set the coordinate system and motion mode for jogging of the active mechanical unit. Information whether the mechanical unit is calibrated or not. Provide name, task, number of axes, active tool and work object and so on. of the mechanical unit. Notify when data of the mechanical unit has changed. Send a start jogging or a stop jogging request to the robot controller.

Continues on next page

## 5 Using the FlexPendant SDK

---

### 5.5.1. ABB.Robotics.Controllers

*Continued*

CAPI domain	Services
RapidDomain	Notification when execution status has changed. Start and stop RAPID execution. Load Rapid programs. Create, read and write RAPID data. Notification when RAPID data or RAPID program has changed. Notification when program pointer has changed. Search for RAPID symbols and so on.
SystemInfoDomain	Information about the active system of the robot controller, for example RobotWare version, system name, release and system paths, existing system options and installed additional options.

---

### Releasing memory

Using CAPI you will create objects that reference unmanaged resources. (For more information, see [Definitions on page 17](#).) It is necessary to explicitly deallocate the memory of such objects by calling their `Dispose` method when they are no longer needed (at application shut down at the latest). Otherwise your application will leak memory, which is a scarce resource on the FlexPendant platform.



#### NOTE!

You may prevent memory leaks and other pitfalls, by studying the chapter [Robust FlexPendant applications on page 165](#).

---

### FlexPendant SDK reference documentation

Although this manual covers a great deal of the FlexPendant SDK functionality, it is by no means complete. The *Reference Manual FlexPendant SDK*, which you open from Windows **Start** button, is the complete reference to the functionality offered by the class libraries.

It also gives valuable code samples and remarks about methods requiring different UAS grants and so on., which is not included in this manual.

## 5.5.2. Accessing the controller

### Overview

Making use of controller functionality requires special attention. Disposal of CAPI objects when closing a view or the entire application is vital to save memory. Another pitfall that must be avoided is updating the user interface on a worker thread.

### Controller instance

Before accessing the controller functionality you need to instantiate the `Controller` object. The best place to do this is normally in the `Install` method, which is called by TAF after the constructor of your view class has executed.

The `Controller` declaration should be done on class scope level:

VB:

```
Private AController As Controller
```

C#:

```
private Controller aController;
```

The recommendation is to instantiate the `Controller` object in the `Install` method:

VB:

```
AController = New Controller
```

C#:

```
aController = new Controller();
```

Using the `Controller` object you can access IO signals, RAPID data variables, event log messages and so on. Most of these objects will be created explicitly and should be disposed of explicitly.



#### NOTE!

Avoid placing any code that may result in an exception before the method `InitializeComponent`. Then at least you have a user interface, where you can display a `MessageBox` with information about what went wrong.



#### NOTE!

It is recommended that if several classes need to access the controller, they all reference the same `Controller` object.

*Continues on next page*

*Continued*

---

#### Subscribing to 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 UpdateOP
AddHandler AController.StateChanged, AddressOf UpdateState
AddHandler AController.MastershipChanged, AddressOf UpdateMast
AddHandler AController.BackupFinished, AddressOf UpdateBack
```

C#:

```
AController.OperatingModeChanged += new
    OperatingModeChangedEventHandler(UpdateOP);
AController.MastershipChanged += new
    MastershipChangedEventHandler(UpdateMast);
Controller.BackupFinished += new
    BackupFinishedEventHandler(UpdateBack);
Controller.StateChanged += new
    StateChangedEventHandler(UpdateState);
```



#### NOTE!

Controller events use their own threads. Study *Controller events and threads on page 50* to find out how to avoid threading conflicts.



#### 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.

---

#### Creating a backup

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. As the backup process is performed asynchronously you can add an event handler to receive a `BackupCompleted` event when the backup is completed.

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
    BackupEventHandler(controller_BackupCompleted);
this.aController.Backup(backupDir);
```



**NOTE!**

There is also a `Restore` method available. The *Reference Manual FlexPendant SDK* is the complete FlexPendant SDK programming guide and is more detailed than this manual. For the `Backup` and `Restore` methods, for example, there are parameter descriptions, remarks, code examples and so on.

**Dispose**

The disposal of the `Controller` object should be done in the `Uninstall` or `Dispose` method of the application view class.

Make a check first that disposal has not already been done. Do not forget to remove any subscriptions to controller events before the `Dispose()` call:

VB:

```
If Not AController Is Nothing Then
    RemoveHandler AController.OperatingModeChanged, AddressOf
        OpMChange
    AController.Dispose()
    AController = Nothing
End If
```

C#:

```
if (aController != null)
{
    aController.OperatingModeChanged -= new
        OperatingModeChangedEventHandler(OpMChange);
    aController.Dispose();
    aController = null;
}
```

**CAUTION!**

VB programmers should be aware that it is a bit tricky to use `WithEvents` together with the `Dispose` pattern on the .NET platform. When you use `withEvents` the .NET framework automatically removes any subscriptions when the object is set to `Nothing`. If you look at the code preceding sample, which does NOT use `WithEvents`, you will understand why such behavior causes problems. When the controller reference is set to `Nothing` and the .NET framework tries to remove its subscription, the internal controller object has already been removed by the `Dispose` call in the preceding statement, and a `NullReferenceException` is thrown. This is not specific to the FlexPendant SDK, but a Microsoft issue. To avoid it you are advised to use `AddHandler` and `RemoveHandler` like in the example.

## 5.5.3. Rapid domain

### 5.5.3.1. Working with RAPID data

---

#### Overview

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

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



#### TIP!

A convenient and user-friendly way to enable the end-user of your application to read and write to specific RAPID data is using the standard FlexPendant Program Data view. For more information, see [Using standard dialog box to modify data on page 114](#).



#### TIP!

Using databinding for RAPID data is a quick way of implementing access to RAPID data. For more information, see [Data of RAPID data and IO signals on page 105](#).

---

#### 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. For more information, see [SearchRapidSymbol method on page 139](#).

#### 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 RAPID data instance “reg1” in the USER module.

VB:

```
Dim Rd As RapidData = Me.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");
```

*Continues on next page*

*Continued*

#### Accessing data declared in a shared module

If your application is to be used with a multi-move system (a system with 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.

VB:

```
Dim Rd As RapidData = Me.AController.Rapid.GetRapidData("reg100")
```

C#:

```
RapidData rd = aController.Rapid.GetRapidData("reg100");
```

Another possibility is using the `Task` object to access the RAPID instance, like this:

```
Task tRob1 = aController.Rapid.GetTask("T_ROB1");
```

```
RapidData rData = tRob1.GetRapidData("reg100");
```



#### **NOTE!**

If `GetRapidData` is called from `Rapid` the RAPID data will be found even if the *-Shared* module is configured to be hidden.



#### **NOTE!**

If the RAPID data does not exist, the return value is `Nothing/null` and an `ArgumentNullException` is thrown. A null check should be performed before trying to use the object.

## 5 Using the FlexPendant SDK

---

### 5.5.3.1. Working with RAPID data

*Continued*

---

#### 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`. The following code explains how this is done, if you want 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 RapidDomain.Bool
Dim rd As RapidData = Me.AController.Rapid.GetRapidData( "T_ROB1",
    "MainModule", "flag" )
'test that data type is correct before cast
If TypeOf rd.Value Is RapidDomain.Bool Then
    rapidBool = DirectCast(rd.Value, RapidDomain.Bool)
'check if the value of the RAPID data is true
If (rapidBool.Value) Then
    ' Do something...
End If
End If
```

C#:

```
//declare a variable of data type RapidDomain.Bool
RapidDomain.Bool rapidBool;
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 only want to read this variable you can use this 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());
```

*Continued*

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

VB:

```
Dim ATool As ToolData
If 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 boolValue = rapidBool.ToString();
```

The string is formatted according to the same principle as described in the following section [IRapidData.FillFromString method on page 126](#).

Here is an example of a more complex data type. The `ToolData.Tframe` property is of the `Pose` type. 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();
```

*Continues on next page*

*Continued*

---

#### 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. An example is the easiest way of illustrating this.

Example:

The `RapidDomain.Pose` structure corresponds to 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;
}
```

This 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.

As you see, the `RapidDomain.Pose` structure consists of two other structures, *trans* and *rot*. The *trans* structure consists of three floats and the *rot* structure 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]]".

This piece of code 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, 0.0, 0.0, 0.0][10, 20, 30]]")
    rd.Value = rapidPose
End If
```

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!**

Using the same principle arbitrarily long RAPID data types can be represented.

**NOTE!**

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

**Writing to RAPID data**

Writing to RAPID data is only possible using the type cast `RapidData` value, to which the new value is assigned. To transfer the new value to the RAPID data in the controller you must finally 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 124](#).

VB:

```
'Assign new value to .Net variable
rapidBool.Value = False
'Write the new value to the data in the controller
rd.Value = rapidBool
```

C#:

```
//Assign new value to .Net variable
rapidBool.Value = false;
//Write to new value to the data in the controller
rd.Value = rapidBool;
```

This was an easy example, 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.

The string must be formatted according to the principle described in the previous 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 Pos = New Pos
APos.FillFromString(" [2,3,3] ")
Me.ATool.Tframe.Trans = APos
Me.Rd.Value = Me.ATool
```

```
C#:
Pos aPos = new Pos();
aPos.FillFromString(" [2,3,3] ");
this.aTool.Tframe.Trans = aPos;
this.rd.Value = this.aTool;
```

**NOTE!**

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

## 5 Using the FlexPendant SDK

---

### 5.5.3.1. Working with RAPID data

*Continued*

---

#### 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. Note, however, that this only works for **persistent** RAPID data.

#### Add subscription

The following code explains how you add a subscription to the `ValueChanged` event:

VB:

```
Addhandler Rd.ValueChanged, AddressOf Rd_ValueChanged
```

C#:

```
this.rd.ValueChanged += rd_ValueChanged;
```

#### Handle event

Implement 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 (AddressOf UpdateGUI), sender, e)  
End Sub
```

C#

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

For more information to learn more about potential threading conflicts in FlexPendant SDK applications, see [Controller events and threads on page 50](#).

#### Read new value from controller

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();  
}
```

© Copyright 2010 ABB. All rights reserved.

*Continues on next page*





#### **NOTE!**

Subscriptions work only for RAPID data declared as PERS.

---

#### **RapidData disposal**

Always dispose of `RapidData` objects when they are no longer needed. If you want to reuse a `RapidData` object, you should make sure that you dispose of the current object first.

VB:

```
If Not Rd Is Nothing Then
    Rd.Dispose()
    Rd = Nothing
End If
```

C#:

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

#### 5.5.3.2. Handling RAPID arrays

---

##### Overview

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

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



##### TIP!

A convenient and user-friendly way of reading and writing array elements is using the standard Program Data view of the FlexPendant. You provide the element you want to have displayed as argument, and the user can view or manipulate the item the way it is usually done on the FlexPendant. For more information, see [Using standard dialog box to modify data on page 114](#).

---

##### ArrayData object

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

VB:

```
Dim RD As RapidData = AController.Rapid.GetRapidData("T_ROB1",  
    "User", "string_array")  
If RD.IsArray Then  
    Dim AD As ArrayData = DirectCast( RD.Value,ArrayData)  
    .....  
End If
```

C#:

```
RapidData rd = aController.Rapid.GetRapidData("T_ROB1", "User",  
    "string_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
Dim aNum As Num
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(1) - 1
        For j As Integer = 1 To ad.Length
            aNum = DirectCast(ad[i, j], Num)
            aSum += Cdbl(aNum)
        Next
    Next
Else
    For i As Integer = 1 To ad.GetLength(1) - 1
        For j As Integer = 1 To ad.GetLength(2) - 1
            For k As Integer = 1 To ad.GetLength(3)
                aNum = DirectCast(ad(i, j, k), Num)
                aSum += Cdbl(aNum)
            Next
        Next
    Next
End If
```

C#:

```
double aSum = 0d;
Num aNum;
if (ad.Rank == 1) {
    for (int i = 1; i <= ad.Length; i++)
    {
        aNum = (Num)ad[i];
        aSum += (double)aNum;
    }
}
else if (ad.Rank == 2)
```

*Continued*

```
{
  for(int i = 1; i < ad.GetLength(1); i++)
  {
    for (int j = 1; j <= ad.Length; j++)
    {
      aNum = (Num)ad[i,j];
      aSum += (double)aNum;
    }
  }
}
else {
  for(int i = 1; i < ad.GetLength(1); i++)
  {
    for(int j = 1; j < ad.GetLength(2); j++)
    {
      for (int k = 1; k <= ad.GetLength(3); 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 aSum As Double = 0R
Dim aNum As Num
For Each aNum As Num In AD
  aSum += Cdbl(aNum)
Next
```

C#:

```
double aSum = 0d;
Num aNum;
foreach(Num aNum in ad)
{
  aSum += (double)aNum;
}
```

### 5.5.3.3. 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 RAPID data 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. The following example retrieves the second *Num* value in the first array of the RAPID data variable referenced by `rd`.

VB:

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

C#:

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

---

#### 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 Num = 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.



#### 5.5.3.4. UserDefined data

---

##### Overview

You often work with RECORD structures in RAPID code. To handle these unique data types, a `UserDefined` class has been implemented. This class has properties and methods to handle individual components of a RECORD.

In some cases implementing your own .NET 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 the path to the declaration of the RAPID data type is passed as argument.

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

VB:

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

C#

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



##### NOTE!

If the module where the RECORD is defined is configured as *-Shared* you only provide the name of the data type as argument, like this:

```
rdt = this.controller.Rapid.GetRapidDataType("processdata");
```

---

**Reading UserDefined data**

You can use a `UserDefined` object 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 UserDefined = DirectCast(rd.Value, UserDefined)
Dim No1 As String = processdata.Components(0).ToString()
Dim No2 AS String = processdata.Components(1).ToString()
```

C#:

```
UserDefined 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 126](#).

---

**Writing to UserDefined data**

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

VB:

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

C#:

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

For more information and code samples, see [IRapidData.FillFromString method on page 126](#) and [Writing to RAPID data on page 127](#).

## 5 Using the FlexPendant SDK

---

### 5.5.3.4. UserDefined data

*Continued*

---

#### 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 `UserDefined` type.

##### Creating ProcessData type

VB:

```
Dim rdt As RapidDataType = Me.ARapid.GetRapidDataType("T_ROB1",  
    "MyModule", "processdata")  
Dim p As ProcessData = New ProcessData(rdt)  
p.FillFromString(rd.Value.ToString())
```

C#

```
RapidDataType rdt = this.aRapid.GetRapidDataType("T_ROB1",  
    "MyModule", "processdata");  
ProcessData p = new ProcessData(rdt);  
p.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  
    End Get  
  
    Set(ByVal Value As UserDefined)  
        data = Value  
    End Set  
End Property  
.....  
End Structure
```

© Copyright 2010 ABB. All rights reserved.

*Continues on next page*



*Continued*

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**

The following 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)
    End Sub

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

```

*Continues on next page*

*Continued*

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

---

### 5.5.3.5. 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.

---

#### SearchRapidSymbol 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.

## 5 Using the FlexPendant SDK

---

### 5.5.3.5. RAPID symbol search

*Continued*

---

#### 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 <code>Block</code> (down) or <code>Scope</code> (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.
SymbolType	Specifies which RAPID type(s) you want to search for. The <code>SymbolTypes</code> enumeration includes <code>Constant</code> , <code>Variable</code> , <code>Persistent</code> , <code>Function</code> , <code>Procedure</code> , <code>Trap</code> , <code>Module</code> , <code>Task</code> , <code>Routine</code> , <code>RapidData</code> . and so on. (Routine includes <code>Function</code> , <code>Procedure</code> and <code>Trap</code> . <code>RapidData</code> includes <code>Constant</code> , <code>Variable</code> and <code>Persistent</code> .)
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.
GlobalRapidSymbol	Specifies whether global symbols should be included.
LocalRapidSymbol	Specifies whether local symbols should be included.
IsInUse	Specifies whether only symbols in use by the loaded RAPID program should be searched.

**Default instance**

`RapidSymbolSearchProperties` has a static method, which returns 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
SymbolType	SymbolTypes.NoSymbol
Recursive	True
GlobalRapidSymbol	True
LocalRapidSymbol	True
IsInUse	True

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

Example:

VB:

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

C#:

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

**NOTE!**

The default instance has the property `SymbolType` set to `NoSymbol`, which means you need to specify it in order to perform a meaningful search.

**NOTE!**

The `SymbolType` property allows you to combine several types in the search. For more information, see the preceding example.

*Continued*

---

#### 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 “robtargt”. 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” instances 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 in the following table:

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.
r   r'	The regular expressions r or r'.
.*	Any character sequence (zero, one or several characters).

*Continues on next page*

*Continued*

#### 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

The following 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.SymbolTypes = SymbolTypes.RapidData  
sProp.LocalRapidSymbol = False  
Dim rsCol As RapidSymbol()  
rsCol = aTask.SearchRapidSymbol(sProp, "num", string.Empty)
```

C#:

```
RapidSymbolSearchProperties sProp =  
    RapidSymbolSearchProperties.CreateDefault();  
sProp.SymbolType = SymbolTypes.RapidData;  
sProp.LocalRapidSymbol = false;  
RapidSymbol[] rsCol;  
rsCol = aTask.SearchRapidSymbol(sProp, "num", string.Empty);
```

*Continued*

---

#### Search for UserDefined RAPID data - example

In the following example an 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.SymbolType = SymbolTypes.RapidData  
Dim rsCol As RapidSymbol()  
rsCol = aTask.SearchRapidSymbol(sProp, "RAPID/T_ROB1/myModule/  
mydata", string.Empty)
```

C#:

```
RapidSymbolSearchProperties sProp =  
    RapidSymbolSearchProperties.CreateDefault();  
sProp.SymbolType = SymbolTypes.RapidData;  
RapidSymbol[] rsCol;  
rsCol = aTask.SearchRapidSymbol(sProp, "RAPID/T_ROB1/myModule/  
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);
```



### 5.5.3.6. RAPID execution

#### Start and Stop RAPID programs

You can start and stop RAPID execution using `Rapid.Start` and `Stop` methods. A `StartResult` from the `Start` method returns the result of the call. Start arguments can be used. `RegainMode` defines how the mechanical unit should handle path status at start. `ExecutionMode` specifies if the program should run continuously, step backward or go to the next `Move` instruction and so on.

The `Stop` method can include a `StopMode` argument, specifying when the program should stop (after current cycle, after completed instruction or immediately).

VB:

```
aController.Rapid.Start (RegainMode.Regain,
                        ExecutionMode.Continuous)
.....
aController.Rapid.Stop (StopMode.Instruction)
```

C#:

```
aController.Rapid.Start (RegainMode.Regain,
                        ExecutionMode.Continuous) ;
.....
aController.Rapid.Stop (StopMode.Instruction) ;
```



#### NOTE!

It is also possible to start a service routine or an ordinary routine without any parameters as if it were a service routine. For more information and code samples, see `Task.CallRoutine` and `Task.CancelRoutine` in the *Reference Manual FlexPendant SDK*.

#### RAPID execution change event

It is possible to subscribe to events that occur when a RAPID program starts to execute and when it stops. This is done on the `Rapid` property of the `Controller` object, like this:

VB:

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

C#

```
aController.Rapid.ExecutionStatusChanged += new
ExecutionStatusChangedEventHandler (UpdateUI) ;
```

For more information and code example on how write the event handlers needed to update the GUI due to a controller event, see [Letting the user know that RAPID data has changed on page 128](#).

*Continued*

---

#### ResetProgramPointer method

The `ResetProgramPointer` method resets the program pointers of all tasks and sets them to the main entry point of the respective task.

VB:

```
aController.Rapid.ResetProgramPointer()
```

C#:

```
aController.Rapid.ResetProgramPointer();
```



#### **NOTE!**

It is also possible to set the program pointer to a specified routine, row or position. For more information and code samples, see `Task.SetProgramPointer` in the *Reference Manual FlexPendant SDK*.

---

### 5.5.3.7. Modifying modules and programs

---

#### Overview

Using the `Task` object it is possible to load and save programs and individual modules. You can also unload programs, get the position of the motion pointer (MP) and the program pointer (PP) as well as modify a robot position.

---

#### Load modules and programs

To load a module or program file you need the path to the file in the file system of the controller. When 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;
}
```

## 5 Using the FlexPendant SDK

---

### 5.5.3.7. Modifying modules and programs

*Continued*

---

#### Save and unload RAPID program

You can save a program using the `SaveProgramToFile` method and unload it using the `DeleteProgram` method. These methods save and unload all modules in the task.

VB:

```
Dim TaskCol As Task() = AController.Rapid.GetTasks()
Dim AnObject As Object
For Each AnObject in TaskCol
    ATask = DirectCast(AnObject, Task)
    ATask.ProgramName = ATask.Name
    ATask.SaveProgramToFile(SaveDir)
    ATask.DeleteProgram()
Next
```

C#:

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

---

#### Save module

You can save a module by using the `Module.SaveToFile` method. As argument you use a path to the controller file system.

VB:

```
AModule.SaveToFile(AFilePath)
```

C#

```
aModule.SaveToFile(aFilePath);
```

---

#### ProgramPointer and MotionPointer

The `Task.ProgramPointer` property returns the current location of the program pointer (module, routine and row number), that is, where the program is currently executing. The same functionality is available for motion pointer by using the `MotionPointer` property.

VB:

```
Dim APP As ProgramPointer = ATask.ProgramPointer
If Not APP = ProgramPointer.Empty Then
    Dim AStartRow As Integer = APP.Start.Row
    .....
```

C#:

```
ProgramPointer pp = aTask.ProgramPointer;
if (pp != ProgramPointer.Empty)
{
    int aStartRow = pp.Start.Row;
    .....
```

© Copyright 2010 ABB. All rights reserved.

*Continues on next page*

---

#### ModifyPosition method

Using the `ModifyPosition` method of the `Task` object you can modify the position of a `RobTarget` instance in the currently loaded program. As arguments you supply a module Name as well as a `TextRange` object. The first `RobTarget` within the text range specified by the `TextRange` object will be changed using the current TCP of the active mechanical unit.

VB:

```
Me.ATask.ModifyPosition(AModule, ATextRange)
```

C#:

```
this.ATask.ModifyPosition(aModule, aTextRange)
```

#### **TIP!**

For more information on `Task` methods and properties, see in the *Reference Manual FlexPendant SDK*.



#### 5.5.4. 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.

---

##### 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. The following example shows a how a signal of type `DigitalSignal` is created:

VB:

```
Dim DISig As DigitalSignal = DirectCast(Signal1, DigitalSignal)
```

C#:

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

The following example shows a how an `AnalogSignal` is created:

VB:

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

C#:

```
AnalogSignal aiSig = (AnalogSignal) signal1;
```

The following example shows a how a `GroupSignal` is created:

VB:

```
Dim GISig As GroupSignal = DirectCast(Signal1, GroupSignal)
```

C#:

```
GroupSignal giSig = (GroupSignal) signal1;
```

##### NOTE!

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



**Getting signals using SignalFilter**

Instead of getting one signal at a time you can use a filter and get a signal collection. 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 =
    controller.IOSystem.GetSignals(aSigFilter)
```

C#:

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

This piece of 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())
    Me.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());
    this.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
```

C#:

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

*Continues on next page*

*Continued*

---

#### Reading IO signal values

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

##### Digital signal

This piece of code reads the digital signal DO1 and checks 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
EndIf
```

C#:

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

##### Analog signal

This piece of code reads the value of the analog signal AO1 and displays it in a textbox:

VB:

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

C#:

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



## Writing IO signal values

In the following example, new values for the IO signals that were retrieved in the previous example are written to the controller.



### 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

This piece of code changes the value of a digital signal in the controller when you check/uncheck a checkbox:

VB:

```
Private Sub checkBox1_Click(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles checkBox1.Click
    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

This piece of 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
```

C#:

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

*Continues on next page*

*Continued*

---

#### 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.

VB:

```
Friend WithEvents aiSig As AnalogSignal
...
AddHandler aiSig.Changed, AddressOf aiSig_Changed
...
Private Sub aiSig_Changed(sender As Object, e As
    SignalChangeEventArgs) Handles aiSig.Changed
End Sub
```

C#

```
this.aiSig.Changed +=new SignalChangeHandler(aiSig_Changed);
```

**NOTE!** The event handler skeleton is auto generated using the Tab key twice after “+=” in the preceding statement:

```
private void aiSig_Changed(object sender, SignalChangeEventArgs 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 aiSig.Changed, AddressOf aiSig_Changed
RemoveHandler aiSig.Changed, AddressOf aiSig_Changed
```

C#:

```
this.aiSig.Changed += new SignalChangeHandler(aiSig_Changed);
this.aiSig.Changed -= new SignalChangeHandler(aiSig_Changed);
```

### 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. 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 ABB.Robotics.Controllers.IOSystemDomain.
    SignalChangedEventHandler(AddressOf UpdateUI), New Object()
    {sender, e})
```

C#:

```
this.Invoke(new ABB.Robotics.Controllers.IOSystemDomain.
    SignalChangedEventHandler(this.UpdateUI), new Object[]
    {sender, e});
```

For more information, see [Controller events and threads on page 50](#).

### Finding out 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
```

C#:

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

#### NOTE!

Do not count on receiving 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.



#### 5.5.5. 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.

---

##### Access 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;
```

---

##### Access 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.

---

### Access 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)
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 msg in cat.Messages)
{
    this.textBox1.Text = msg.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(sender As Object, e As
    MessageWrittenEventArgs) Handles Log.MessageWritten
    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 51](#).

#### 5.5.6. Motion domain

---

##### Overview

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

Using a `MotionSystem` object you can send jogging commands to an mechanical unit and get or set the incremental jogging mode. Using a `MechanicalUnit` object you can get a lot of information about the mechanical units of the robot system.

You can also subscribe to changes of the mechanical unit, for example changed tool, work object, coordinated system, motion mode or incremental step size.

---

##### MotionSystem object

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

VB:

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

C#

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

---

##### 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. Mechanical units are available through the `MotionSystem` object. If only the active mechanical unit is of interest you may use the method `GetActiveMechanicalUnit`.

VB:

```
Dim aMechCol As MechanicalUnitCollection =
    aController.MotionSystem.GetMechanicalUnits()
Dim aMechUnit As MechanicalUnit =
    aController.MotionSystem.GetActiveMechanicalUnit();
```

C#:

```
MechanicalUnitCollection aMechCol =
    aController.MotionSystem.GetMechanicalUnits();
MechanicalUnit aMechUnit =
    aController.MotionSystem.GetActiveMechanicalUnit();
```

---

**Jogging**

It is possible to jog the active mechanical unit using the `SetJoggingCmd` method and the calls `JoggingStart` and `JoggingStop`. Depending on the selected `MotionMode` and `IncrementalMode` different joints and speeds are configured.

VB:

```
aController.MotionSystem.JoggingStop()
aMechUnit.MotionMode = MotionModeType.Linear
aController.MotionSystem.IncrementalMode =
    IncrementalModeType.Small
aController.MotionSystem.SetJoggingCmd(-50, 50, 0)
aController.MotionSystem.JoggingStart()
```

C#:

```
aController.MotionSystem.JoggingStop();
aMechUnit.MotionMode = MotionModeType.Linear;
aController.MotionSystem.IncrementalMode =
    IncrementalModeType.Small;
aController.MotionSystem.SetJoggingCmd(-50, 50, 0);
aController.MotionSystem.JoggingStart();
```

---

**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 aController As New Controller()
Dim aRobTarget As RobTarget =
    c.MotionSystem.ActiveMechanicalUnit.GetPosition(CoordinateSystemType.World)
Dim aJointTarget As JointTarget =
    c.MotionSystem.ActiveMechanicalUnit.GetPosition()
```

C#:

```
Controller aController = new Controller();
RobTarget aRobTarget =
    aController.MotionSystem.ActiveMechanicalUnit.GetPosition(CoordinateSystemType.World);
JointTarget aJointTarget =
    aController.MotionSystem.ActiveMechanicalUnit.GetPosition()
;
```

*Continued*

---

#### DataChanged event

By subscribing to the DataChanged event of the MechanicalUnit object, you will be notified when a change of tool, work object, coordinated system, motion mode or incremental step size occurs.

VB:

```
AddHandler aMechUnit.DataChanged, AddressOf aMech_DataChanged
.....
Private Sub aMech_DataChanged(sender As Object, e As
    MechanicalUnitDataEventArgs)
    Select e.Reason
        Case MechanicalUnitDataChangeReason.Tool
            ChangeOfTool(DirectCast(sender, MechanicalUnit))
        Case MechanicalUnitDataChangeReason.WorkObject
            .....
    End Select
End Sub
```

C#:

```
aMechUnit.DataChanged += new
    MechanicalUnitDataEventHandler(aMech_DataChanged);
.....
private void aMech_DataChanged(object sender,
    MechanicalUnitDataEventArgs e) {
    switch (e.Reason) {
        case MechanicalUnitDataChangeReason.Tool:
            ChangeOfTool((MechanicalUnit) sender)
        case MechanicalUnitDataChangeReason.WorkObject:
            .....
    }
}
```



#### TIP!

Read more about the classes, methods and properties available in the MotionDomain in the *Reference Manual FlexPendant SDK*.



---

## 5.5.7. File system domain

---

### Overview

Using the FlexPendant SDK `FileSystemDomain` you can create, save, load, rename, and delete files on the controller. You can also create and delete directories.

---

### Accessing files and directories

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

VB:

```
Private aFileSystem As FileSystem = aController.FileSystem
```

C#:

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

---

### Controller and FlexPendant file system

You can find and set the remote directory on the controller and the local directory on the FlexPendant device by 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;
```

---

### Loading controller files

You can load a file from the controller to the FlexPendant using the `GetFile` method. An exception is thrown if the operation fails. The arguments are complete paths including filenames.

VB:

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

C#:

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

---

### Saving files

You can save a file to the controller file system by using the `PutFile` method. An exception is thrown if the operation fails. The arguments are complete paths including filenames.

VB:

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

C#:

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

*Continues on next page*

*Continued*

---

#### 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.

The following example uses search pattern to limit the search.

VB:

```
Dim anArray As ControllerFileSystemInfo()  
Dim info As ControllerFileSystemInfo  
anArray = aController.FileSystem.GetFilesAndDirectories("search  
    pattern")  
Dim I As Integer  
For I = 0 To array.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];  
    .....  
}
```

---

#### 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 uses the Wildcard pattern matching of 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)
[ <i>charlist</i> ]	Any single character in <i>charlist</i>
[! <i>charlist</i> ]	Any single character not in <i>charlist</i>

#### TIP!

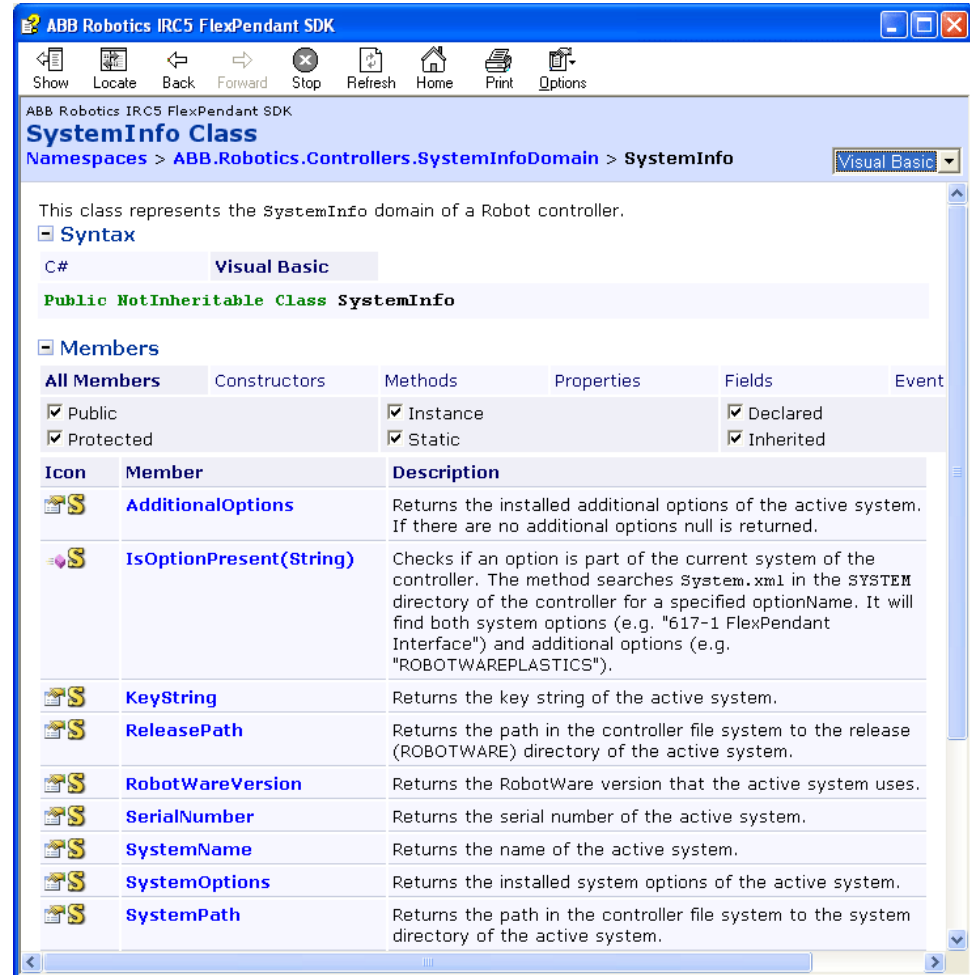
For more information on classes, methods and properties, see `FileSystemDomain` in the *Reference Manual FlexPendant SDK*.



## 5.5.8. System info domain

### Overview

The `SystemInfoDomain` provides information about the active robot system. This is mainly done through the static methods and properties of the `SystemInfo` class:



5.8.1\_1

### SystemInfo class

The functionality of the `SystemInfoDomain` is accessed by calling the static methods of the `SystemInfo` class. The following code explains how to retrieve the path in the controller file system to the release (ROBOTWARE) directory of the active system.

Example:

```
string rWDir =
    ABB.Robotics.Controllers.SystemInfoDomain.SystemInfo.ReleasePath
```

Likewise, the path to the active system directory can be retrieved:

```
string sysDir =
    ABB.Robotics.Controllers.SystemInfoDomain.SystemInfo.SystemPath
```

*Continues on next page*

*Continued*

---

#### System options

Using the `SystemInfo.SystemOptions` property you can retrieve the system options of the currently active robot system. The result is an array of `SystemOption` objects. If you list the `Name` property of these objects you will get the same result as shown in the *SystemBuilder* of *RobotStudio*, e.g:

**Options:**

```
RW Control module key
RobotWare OS and English
644-8 Swedish
  - 645-5 Chinese
603-1 Absolute Accuracy
611-1 Path Recovery
616-1 PC Interface
617-1 FlexPendant Interface
623-1 Multitasking
875-1 DieCast
```

5.5.8\_2

You can retrieve sub options of a system option by using the `SystemOption.SubOptions` property.

---

#### Additional options

Using the `SystemInfo.AdditionalOptions` property you can find out which additional options are installed in the robot system. The result is an array of `AdditionalOption` objects. The `AdditionalOption.Path` property returns the path to the installation directory of the additional option.

The following `AdditionalOption` properties are available:

- `KeyString`
- `Name`
- `Path`
- `Type`
- `VersionInfo`



**NOTE!**

For more information on `SystemInfoDomain` with code examples, see *Reference Manual FlexPendant SDK*.

## 6 Robust FlexPendant applications

### 6.1. Introduction

#### Overview

Developing an application for a device with limited resources, such as memory and process power, can be quite demanding. Moreover, to have an application executing around the clock will reveal weaknesses in design and implementation that may cause slow performance or FlexPendant hangings.

At worst, your application will drain the FlexPendant of all memory during production, and cause an out-of-memory crash. It can even slow down the performance of the robot controller due to excessive use of controller resources.

This chapter describes how to design and implement reliable and well performing applications for the FlexPendant. It presents some good practices to utilize, as well as some pitfalls that should be avoided.

#### Technical overview of the FlexPendant device

The FlexPendant device consists of both hardware and software and is a complete computer in itself, with its own memory, file system, operating system and processor.

It is an integral part of IRC5, connected to the controller by an integrated cable and connector. Using the hot plug button option, however, you can disconnect the FlexPendant in automatic mode and continue running without it.

There are ways to restart the FlexPendant without having to restart the controller (For more information, see [Restart the FlexPendant on page 35](#)). At a FlexPendant restart the assemblies and resources of FlexPendant SDK applications are downloaded to the FlexPendant file system from the robot controller.

FlexPendant applications run on Windows CE, a scalable embedded operating system, and the .NET Compact Framework, which is Microsoft's lightweight version of the .NET Framework, intended for small devices.

This is the size of the FlexPendant touch screen:

FlexPendant screen	Size
Total display	640 * 480 pixels
FlexPendant SDK Application display	640 * 390 pixels

The FlexPendant uses these kinds of memory:

Memory type	Function
Flash - 16 MB	Stores the FlexPendant standard software, the Windows CE operating system in compressed format and the registry.
RAM - 64 MB	At boot time the compressed image is copied to RAM. All execution of code uses RAM.
E <sup>2</sup> PROM	Stores touch screen calibration values, joystick calibration values and so on. Only used internally.

*Continues on next page*

## 6 Robust FlexPendant applications

---

### 6.1. Introduction

*Continued*

---

#### How large can a custom application be?

You may wonder about the maximum size of your custom application. There is no exact answer to that question, as there are many variables to take into account. For a rough estimation the following table can be used. As you see, the operating system uses about 8 MB and the ABB base software about 25 MB. This means that half of the available RAM memory is already used once the FlexPendant has started up. The standard applications of the ABB menu and the FlexPendant SDK applications will all share the memory that is left. As a rule of thumb, about 20 MB should be available for custom applications.

FlexPendant memory resources	
RAM	64 MB
Operating system	8 MB
ABB base software	25 MB
Custom applications	~20 MB

This is some advice to help you make sure your application does not exceed the memory limitation:

1. Do not allow more than ONE instance by setting the `TpsViewType` parameter of the `TpsView` attribute to `Static`. For more information, see [Application type on page 41](#).
2. Avoid excessive use of images. Do not use bigger images than necessary. Check the size of the images your application will use.
3. Use `fpcmd "-memShow"` to check the amount of memory in use when your application is active. Open a couple of Program Editors and start RAPID execution. For more information, see [Discover memory leaks on page 169](#).
4. Avoid `static` data and methods.
5. Release memory for objects that are not used by calling their `Dispose` method.

## 6.2. Memory management

### Garbage collection and Dispose

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 should not have to free memory which has been allocated by the use of `new`. A drawback, when memory is limited, is that the execution of the garbage collector is non-deterministic. There is no way of knowing exactly when garbage collection will be performed.

The `IDisposable` interface, however, represents a way to obtain deterministic deallocation of resources. You should therefore call `Dispose()` on all disposable object when they are no longer needed, as this will free up valuable resources as soon as possible.

Moreover, objects used to access robot controller resources, must be released by the custom application by an explicit call to their `Dispose` method. `SignalBindingSource` and `RapidDataBindingSource` objects, as well as all other objects located in the components pane of the Visual Studio Designer must also be explicitly disposed of, or else your application will have a permanent memory leak.

#### NOTE!

The creator of an object implementing the `IDisposable` interface is responsible for its lifetime and for calling `Dispose`.



#### TIP!

You may wonder why the .NET garbage collector cannot ensure that all objects no longer referenced are finally destroyed? The FlexPendant development team have tried hard to remove any remaining objects of an SDK application at application shut down, for example implemented finalizers for the `TpsControl`, `RapidDataBindingSource` and `SignalBindingSource` classes. Due to Microsoft's implementation, however, the .NET runtime nonetheless refuses to destroy these objects unless their `Dispose` method is called. This behavior is under debate. If you are curious to find out more about this, these community articles may be of interest.

#### **Dispose, Finalization, and Resource Management** (Joe Duffy):

<http://www.bluebytesoftware.com/blog/PermaLink.aspx?guid=88e62cdf-5919-4ac7-bc33-20c06ae539ae>

#### **Garbage Collection: Automatic Memory Management in the Microsoft .NET**

**Framework** (MSDN-magazine): <ms-help://ms.msdnqtr.v80.en/ms.msdn.v80/dnmag00/html/GCI.htm>

**Finalization** - cbrumme's WebLog: <http://blogs.msdn.com/cbrumme/archive/2004/02/20/77460.aspx>

## 6 Robust FlexPendant applications

### 6.2. Memory management

Continued

#### Application Framework usage - ITpsViewSetup

The application framework TAF, which hosts the controls that make up a FlexPendant application, offers some mechanisms that should be used by client applications. For more information about TAF, see *Understanding the FlexPendant application life cycle on page 37*.

Your application view class should implement `ITpsViewSetup` and `ITpsViewActivation`. For general information on these interfaces, see *ITpsViewSetup and ITpsViewActivation on page 43* and to learn how to use `ITpsViewActivation` to improve performance, see *Application Framework usage - ITpsViewActivation on page 171*.

The `ITpsViewSetup` interface has two methods: `Install` and `Uninstall`. `Install` is called when the view is being created, right after the constructor has been executed.

`Uninstall` is called when the client view is closed down. After `Uninstall` has been called, TAF will also call the `Dispose` method of the view class. These methods thus offer the last opportunity for you to clean up and release memory and system resources held by the custom application.

#### NOTE!

You should close down the application by using the close button, [x], in the upper right corner of the display, in the same way as the Program Data or other standard applications are closed. Never implement a close button on the first view. When the application is closed the correct way, first `Deactivate`, then `Uninstall` and finally `Dispose` will be called by TAF.



#### How to program the Dispose method - example

When starting a new FlexPendant project in Visual Studio a skeleton for the `Dispose` method is auto generated. You should use it to add code for cleaning up as shown in the following screenshot:

```
/// Releases all resources used by this instance.
/// </summary>
/// <param name="disposing"><b>true</b></param> releases both managed and unmanaged resources (called by Dispose)
/// <param name="disposing"><b>false</b></param> releases only unmanaged resources (called by finalizer).</param>
protected override void Dispose(bool disposing)
{
    if (!this.IsDisposed)
    {
        if (disposing == true)
        {
            /*TODO: Dispose of all CAPI objects.
            * Remember to remove any subscriptions first.*/
            if (_controller != null)
            {
                _controller.OperatingModeChanged -= new ABB.Robotics.Controllers.OperatingModeChangedE
                _controller.Dispose();
                _controller = null;
            }

            /*TODO: Add code to dispose of all GUI components that have not been added
            *to the controls collections like this: this.Controls.Add(this.comboBox1).
            * (These will be disposed of by the base class.*/
            if (this.alphaPad1 != null)
            {
                this.alphaPad1.Dispose();
                this.alphaPad1 = null;
            }

            /*TODO: Add code to dispose of System.Drawing objects such as Bitmap, Pen, Brush etc*/
        }
    }
    /*Always call Dispose of base class*/
    base.Dispose(disposing);
}
```

7.2\_1

© Copyright 2010 ABB. All rights reserved.

Continues on next page



**NOTE!**

Error handling should be added to the `Dispose` method (left out in the figure).

**NOTE!**

Ensure you have unsubscribed to any events from the robot controller before you call the `Dispose` method of a CAPI object. If it has been done in the `Deactivate` method, which is what is usually recommended, you should not do it again in the `Dispose` method. Also ensure you do not try to access an object after it has been disposed.

**Discover memory leaks**

When your application interacts with the robot controller, unmanaged objects are created under the hood. If you forget to call `Dispose` on such objects there will be memory leaks.

You are recommended to test the memory consumption of your application. Use a controller console window on your PC and the command `fpcmd_enable_console_output 3` to enable printouts. Then use the `"-memShow"` command with a period argument that produces a printout every 500 second for example, like this:

```
-> period 500, fpcmd, "-memShow"
```

For more information on console window, see [Serial connection to the Console port on page 24](#)

*Result:*

```
task spawned: id = 0xba7f3b8, name = t2value = 195556280 = 0xba7f3b8-
> [fp]: Available memory: 20881408 (tot 40394752), disk: 737148 (tot
1728512) load: 54(261955) [fp]:
```

The load component indicates the amount of used RAM memory on the FlexPendant (expressed in percentage of the total amount) and the number in the parenthesis indicates the time taken in *ms* after start-up. For example in the preceding *Result*, 54 is the percentage of used memory in the FlexPendant and 261955 *ms* is the time taken after start-up.

Test all functions of your application with several other FlexPendant views visible in the task bar and possibly one or several RAPID tasks executing. Observe how your implementation affects memory. Close your application and make sure the same amount of memory is available as before opening it.

The following procedure shows yet another way of checking that your application cleans up correctly:

Step	Action
1	Log out from the FlexPendant by clicking <b>Log Off</b> on the ABB menu.
2	In the controller console window write <code>fpcmd_enable_console_output</code> .
3	Write <code>fpcmd "-a"</code> .
4	For each cpp class check <i>Number of instances</i> . It shall be 0, except for <code>AdaptController (1)</code> and <code>AdaptEventImpl (1)</code> .
5	If the previous step shows that there are unmanaged objects left, you need to search your code for missing <code>Dispose</code> calls.

**NOTE!**

You must use the real FlexPendant device when verifying memory consumption and looking for memory leaks in your code.

## 6 Robust FlexPendant applications

---

### 6.3. Performance

### 6.3. Performance

---

#### About performance

A FlexPendant application cannot meet hard real time demands. There should however be no difference in performance between a FlexPendant SDK application and the standard applications of the device. If your application is slow, the advice in this section will be useful.



#### TIP!

Do not get overwhelmed by the number of restrictions presented in this section. Most of the time you will not notice any difference if you decide to neglect a few of them. Yet - it is good to know what can be done whenever performance does become an issue.

---

#### Less code means faster code

A general piece of advice (maybe too obvious a recommendation) is that less code normally means faster code. Remember that methods that are executed frequently, such as `OnPaint`, event handlers and so on, must be efficient. Everything that can be computed once and stored for future use should be computed only once.

---

#### Fewer controller accesses means faster code

The best thing you can do to improve performance is probably to ensure that you do not access controller functionality more often than necessary. To achieve this you need to understand CAPI, the SDK class libraries used to access robot controller functionality.

It is very easy to use a property of a class and not realize that an access to the controller is actually made. A general recommendation is to try to use subscriptions to controller information (events) where applicable, and let your application store updated values instead of performing numerous reading operations toward the same controller resource.



#### TIP!

You can easily get an estimate of the number of controller accesses your application performs in different user scenarios by using a robot controller console. The console command `robdisp_watch 2` monitors and prints controller accesses made by the FlexPendant to the console. Click a button of your application for example, and study how the FlexPendant and the controller communicate in response to this action. Enter the command `robdisp_watch 0` to turn this service off. (The monitor service slows down the response time of your application, due to printing to the console. So do not get worried if your application seems unusually slow!)



#### NOTE!

Keep in mind that an excessive number of controller accesses will slow down the performance of your application. At worst, it may even affect the performance of the robot controller.



#### NOTE!

Excessive use of subscriptions may also be a problem. If your application has to handle a continuous load of Rapid data and I/O signal events, and your event handlers need to manipulate the data before presenting it, GUI interaction may become slow. As a rule of thumb, do not estimate more than 20 events/sec. and keep the event handlers thin.

---

#### Fewer objects means faster code

Fewer objects means better performance. If you know that an object will be used again, create it once and keep a reference. Also, try not to create objects that may not be used.

Reusing existing objects instead of creating new ones is especially important when executing the same code repeatedly, for example in tight loops.

The following example shows how this should be done:

```
object _o = new object();
for (int i = 0; i < 100000; i++)
{ ... }
```



#### TIP!

Do not create several `Controller` objects, but reuse it by sending it as a parameter when creating a new view, or share it between classes as a public property.

---

#### Transferring files

Transferring files between the device and the robot controller takes time and also occupies storage memory on the device. Write efficient and fault-tolerant code if it must be done.

---

#### Application Framework usage - `ITpsViewActivation`

The `ITpsViewActivation` interface is used by TAF to hide and display your application. It has two methods: `Activate` and `Deactivate`. The former is called by TAF at the creation of the client view, right after the `ITpsViewSetup.Install` method has been executed. The latter is called at shut down of the client view, right before the `ITpsViewSetup.Uninstall` method is executed.

The interface is also used when you select another application on the task bar. TAF then calls `ITpsViewActivation.Deactivate` (but not `ITpsViewSetup.Uninstall`). Likewise, when the application regains focus via the task bar icon, TAF calls `ITpsViewActivation.Activate` (but not `ITpsViewSetup.Install`).

It is recommended to enable and disable any subscriptions to controller events in the `ITpsViewActivation` methods, as valuable resources should not be held when the application is not used. Note that current values should be read before enabling the subscription.

For the same reason, any timers can be activated and deactivated in these methods. That way timers will not run when other applications are in focus, thus saving processor power.

## 6 Robust FlexPendant applications

---

### 6.3. Performance

*Continued*

---

#### Excessive string manipulation is costly

The string class is an immutable type, that is, once a string is created its value cannot be changed. This means that string methods that seem to modify the string in fact create new strings.

Look at a this string concatenation example:

```
string name = "192.168.126.1";  
string str = "/" + name + "/" + "RAPID";
```

Four strings will be appended to one resulting string. No less than four additional strings will be created and allocated when the right hand side of the assignment is executed. A better idea is to use the `StringBuilder` class or `string.Format` (which uses the `StringBuilder` internally):

```
string str = string.Format("/{0}/RAPID", name);
```

As a rule, if you are going to do only one string operation on a particular string, you can use the appropriate string method. It is when you start doing numerous string operations that you need to use `StringBuilder` or `string.Format`.

---

#### Avoid Control.Refresh

`Control.Refresh()` must only be used when an immediate update of the GUI is absolutely required. `Refresh` makes a direct call to the `OnPaint` method of the `Control`, and is therefore much more costly than `Control.Invalidate()`.

Several calls to `Invalidate` will not mean several calls to `OnPaint`. When the GUI thread processes the `Invalidate` message all queued up messages of the same control are handled at the same time, thus saving process power.



#### NOTE!

In the 2.0 version of .NET CF there are new methods to be used for GUI controls inheriting `System.Windows.Forms.Control`:

- `SuspendLayout()`
- `ResumeLayout()`
- `BeginUpdate()`
- `EndUpdate()`

These methods are used to control GUI updates while modifying a GUI element. The control will not be drawn when `SuspendLayout` has been called, for example, but is blocked until `ResumeLayout` is called.

---

#### Avoid boxing and unboxing

A common reason for slow code is unintentional boxing and unboxing. To avoid this, you need to be aware of the difference between *reference* and *value* types. Reference types (the keyword *class* is used) are always allocated on the heap, while value types are allocated on the stack; unless embedded into a reference type.

Boxing is the operation where a value type, is converted to a reference type. It is done automatically when a reference to a value type is required. Then a new object will be created, allocated on the heap with a copy of the original data.

Here are some examples of not so obvious boxing/unboxing:

- Using the `foreach` statement on an array that contains value types will cause the values to be boxed and unboxed.
- Accessing values of a `Hashtable` with a value type key, will cause the key value to be boxed when the table is accessed.
- Using an `ArrayList` with value types; this should be avoided - use typed arrays instead. Typed arrays are also better because of type safety, as type checking can be performed at compile time.
- The following methods should be overridden in order to avoid unnecessary boxing/unboxing: `Equals()`, `GetHashCode()`.

---

#### Foreach

Using a `for` loop is often faster than using the `foreach` statement, especially if a large number of iterations are made. The reason is that the JIT compiler (see [About terms and acronyms on page 17](#)) is prohibited to optimize the code execution when `foreach` is used.

However, `foreach` makes the code more readable and is therefore a better option when performance is not crucial.

---

#### Reflection is performance demanding

Reflection is a mechanism used to read the meta data of an assembly. The `typeof` operator, for example, uses reflection to determine the type of an object. Another example is `object.ToString()`, which also uses reflection.

As reflection is very performance demanding you are recommended to override or to avoid the `ToString()` method for reference types.

---

#### Efficiently parsing Xml

`XmlTextReader` and `XmlDocument` can both be used to parse xml data. The `XmlTextReader` is the preferred option in most cases; it is more light weight (less memory footprint) and a lot faster to instantiate. The limitation of the `XmlTextReader` is that forward only reading is possible.

The xml structure may also have an impact on performance. If xml data is organized in non-flat way, search operations will be faster, as a large portion of the information can be skipped. This is achieved by using categories and sub categories.

## 6 Robust FlexPendant applications

---

### 6.4. Reliability

## 6.4. Reliability

---

### Overview

This section presents information on how to make the application robust. The most important mechanism related to robustness and reliability is error handling. Avoiding thread conflicts and memory leaks are however also important means of improving reliability.

---

### Error handling in .NET applications

As already pointed out in this manual, Microsoft recommends that exceptions are used to discover and report anomalies in .NET applications. If you are not sure about when to use exceptions or how to do the implementation you should read *Exception handling on page 54* to understand the general idea before moving on to the FlexPendant specific information of this section.

---

### SDK exception classes

The `ABB.Robotics` namespace provides quite a few exception classes, used by the FlexPendant SDK and SDK applications. In addition, the SDK also uses `System` exceptions and may throw a `System.ArgumentException` object for example.

Whenever an exception is thrown, your application must catch it and take proper action. You catch an exception by using the `try-catch(-finally)` statements. For more information about how to implement these statements, see *Exception handling on page 54*.

As you see in the following image `GeneralException` derives from `BaseException`, which in turn derives from `System.ApplicationException` of the .NET Framework class library.

`ApplicationException` is thrown by user programs, such as the FlexPendant SDK, not by the Common Language Runtime (CLR, for more information, see *Definitions on page 17*). It therefore represents a way to differentiate between exceptions defined by applications versus exceptions defined by the system.

ABB Robotics IRC5 FlexPendant SDK

**GeneralException Class**

Common exceptions to be handled by caller.

For a list of all members of this type, see [GeneralException Members](#).

[System.Object](#)

[System.Exception](#)

[System.ApplicationException](#)

[ABB.Robotics.BaseException](#)

**ABB.Robotics.GeneralException**

[Derived types](#)

```
public class GeneralException : BaseException
```

**Thread Safety**

Public static (**Shared** in Visual Basic) members of this type are safe for multithreaded operations. Instance members are **not** guaranteed to be thread-safe.

**Requirements**

**Namespace:** [ABB.Robotics](#)

**Assembly:** ABB.Robotics (in ABB.Robotics.dll)

**See Also**

[GeneralException Members](#) | [ABB.Robotics Namespace](#)

7.4\_1

As you see there are derived types under `GeneralException`. The following table lists and briefly describes these types:

ABB Robotics IRC5 FlexPendant SDK

**ABB.Robotics Namespace**

[Namespace hierarchy](#)

**Classes**

Class	Description
<a href="#">BaseException</a>	ABB.Robotics exception base class.
<a href="#">FPBase</a>	FlexPendant object base
<a href="#">GeneralException</a>	Common exceptions to be handled by caller.
<a href="#">InternalException</a>	Internal error exception.
<a href="#">MasterRejectException</a>	The user does not have required mastership for the operation.
<a href="#">ModeRejectException</a>	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.
<a href="#">ModPosException</a>	The modify position operation failed. The failing task name and reason are available.
<a href="#">RejectException</a>	Operation rejected by the controller safety access restriction mechanism.
<a href="#">ResourceHeldException</a>	Requested resource is held by someone else.
<a href="#">SysfailRejectException</a>	The operation is not allowed in SysFail.
<a href="#">UasRejectException</a>	The user is denied access to the operation.

7.4\_2

**TIP!**

For more information on FlexPendant SDK exception classes, see *Reference Manual FlexPendant SDK*.

*Continues on next page*

## 6 Robust FlexPendant applications

---

### 6.4. Reliability

*Continued*

---

#### Thread affinity

You should be aware that execution of code modifying the user interface, has to be done on the thread that created the GUI control, the so called GUI thread.

An application initially starts with a single thread. Normally all user interface controls are created by this thread. Windows CE user interface objects are characterized by thread affinity, which means that they are closely coupled with the thread that created them.

Interacting with the message queue of an interface control from a thread other than the creating thread may cause data corruption or other errors. This restriction applies to the thread pool as well as to explicitly created threads.

When executing on a secondary thread, a so called worker thread, an update of the user interface must therefore be done very carefully, enforcing a switch to the GUI thread. This is in fact a very common scenario, as controller events use their own threads and should often be communicated to the end user by a GUI update.



#### NOTE!

Thread affinity is especially relevant as for robot controller events, as these by default execute on a background thread. For more information and code samples, see [GUI and controller event threads in conflict on page 50](#) and [Invoke method on page 51](#). The following section also deals with the same issue.

---

#### Invoke

In order to execute a method on the GUI thread `Control.Invoke` can be used. It should however be done carefully, as it makes a synchronous call to the specified event handler, which blocks execution until the GUI thread has finished executing the method. All concurrent calls to `Invoke` will be queued and executed in their queue order by the GUI thread. This could easily make the GUI less responsive.

There is now an asynchronous version of `Invoke`, which should be used instead whenever possible. `TpsControl.BeginInvoke` is a non blocking method, which lets the worker thread continue execution instead of waiting for the GUI thread to have processed the method

Remember that `Invoke` as well as `BeginInvoke` should only be used on code that modifies the user interface. You should keep the execution on the background thread as long as 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`.

---

#### Memory leaks

As FlexPendant applications are supposed to run without interruption, no memory leaks can be permitted. It is your responsibility to properly clean up and call `Dispose`. Take your time studying [How to avoid memory leaks on page 77](#), [Memory management on page 167](#) and [Technical overview of the FlexPendant device on page 165](#).



---

### Utilizing multi-threading

Threading enables your program to perform concurrent processing so you can do more than one operation at a time. For example, you can use threading to monitor input from the user, and perform background tasks simultaneously. The `System.Threading` namespace provides classes and interfaces which enable you to easily perform tasks such as creating and starting new threads, synchronizing multiple threads, suspending threads and aborting threads.

The classes `Thread` and `ThreadPool` can be used to execute methods on a worker thread. Use `ThreadPool` for temporary usage of a background thread when a task is meant to terminate fairly soon. Use `Thread` only for background work that will persist, for example a thread that fetches data from the controller continuously.

There are two timers available in .NET CF that can be used to execute methods periodically at specified intervals. There is however, an important difference between these. The `System.Threading.Timer` executes the method on a background thread, while the `System.Windows.Forms.Timer` executes the method on the UI thread.

**NOTE!**

Use `System.Threading.Timer` if you want to poll data from the controller in order to reduce the load on the GUI thread.

---

### Lock statement

The `lock` statement is used in multi-threaded applications to make sure access to a part of the code is made by one thread exclusively. If a second thread attempts to lock code which has already been locked by another thread, it must wait until the lock is released.

Remember to limit the code segment that you want to control, that is, only lock what is absolutely necessary to make the code thread safe:

```
Object thisLock = new Object();
lock (thisLock)
{
    // Critical code section
}
```

Deadlocks must be avoided. They can occur if more than one lock is used. If more than one lock object must be held, they must always be locked and released in the same order, wherever they are used.

**NOTE!**

If a deadlock occurs, the FlexPendant system will hang. If this happens you should attach the device to the Visual Studio debugger and study the call stack of the threads in the *Threads* window.

**CAUTION!**

Using the `lock` statement in combination with a call to `Invoke` is a potential cause to deadlock situations, since `Invoke` is a blocking call. In short, be careful if you use locks!

*Continues on next page*

## 6 Robust FlexPendant applications

---

### 6.4. Reliability

*Continued*

---

#### Multicast delegates

If a multicast delegate is executed, the execution of the delegates is terminated if an exception is thrown by one of the delegates. It means that the remaining delegates in the list will not be executed if an exception is thrown. This situation can cause erratic behavior, which may be tricky to trace and debug.

Therefore, if you have numerous delegates which are to execute as the result of the same event, you may want to implement the `GetInvocationList` method. It retrieves a list with a copy of the delegates. This list can be iterated and each delegate called directly:

```
private void OnChange()
{
    Delegate[] evtHandlers = null;

    lock (_lockobj)
    {
        if (_changeHandler != null)
            evtHandlers = _changeHandler.GetInvocationList();
    }

    if (evtHandlers != null)
    {
        for (int i = 0; i < evtHandlers.Length; i++)
        {
            Delegate d = evtHandlers[i];
            try
            {
                ((EventHandler)d)(this, new EventArgs());
            }
            catch (System.Exception e)
            {
                ABB.Robotics.Diagnostics.Debug.Assert(false,
                    string.Format("Exception in OnChange: {0}", e.ToString()));
                ABB.Robotics.Diagnostics.Trace.WriteLine
                    (string.Format("Exception in OnChange: {0}", e.ToString()));
            }
        }
    }
}
7.4_3
```

# 7 FlexPendant - Debugging and troubleshooting

## 7.1. Debugging

### Introduction

Using the Visual Studio debugger for a FlexPendant 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.

*Continues on next page*

## 7 FlexPendant - Debugging and troubleshooting

### 7.1. Debugging

*Continued*

Code	Description
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.
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.

© Copyright 2010 ABB. All rights reserved.

*Continues on next page*

*Continued*

Code	Description
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.
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.

*Continues on next page*

## 7 FlexPendant - Debugging and troubleshooting

---

### 7.1. Debugging

*Continued*

Code	Description
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.
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.

## 7.2. Debug output

### Overview

It is possible to get debug output from the FlexPendant by using the network. This is useful for several reasons. It will reveal any exceptions thrown during execution, for example, providing you with error messages and call stacks. Moreover, it can help you check memory consumption and memory leaks in your application.

The FlexPendant message is packed into a network message and sent out on the network as a broadcast message on port 9998. Such a message can be picked up in different ways:

- Connect a hub on the local FlexPendant network and connect a PC to it. Use *nc.exe* to pick up the messages (`nc -lup 9998`).
- The messages sent are also stored in a ring-buffer of size 100kB. To read the buffer to file and store it on the controller, you can use the FlexPendant Command Server. Use *fpcmd* “-d”.
- It is also possible to start a task on the controller that listens to port 9998 and displays all messages in the console buffer. Use command *fp\_enable\_console\_output*.



#### TIP!

For a list of exceptions that the IRC5 Controller may throw, see [Exception error codes on page 179](#).

### Enable debug output

To enable debug output write `fp_enable_console_output` in the controller console window. For more information on console window, see [Serial connection to the Console port on page 24](#).

These are the console commands used to enable and disable printouts:

Console command	Result
<code>fp_enable_console_output 1</code>	Starts producing printouts from RobotWare to the robot controller console.
<code>fp_enable_console_output 2</code>	Starts producing printouts from SDK application.
<code>fp_enable_console_output 3</code>	Combines the two preceding: RobotWare + SDK printouts.
<code>fp_disable_console_output</code>	Stops printout to the robot controller console.

The following command can be used to retrieve detailed status of the robot controller, which may be useful, although it may not be specifically related to your application.

Console command	Result
<code>fpcmd “-d”</code>	Produces a log file with extensive information on system status to the robot controller file system( <code>hd0a/temp</code> ). Use an ftp client or the <i>File Manager</i> in RobotStudio to transfer the file to your PC.

*Continues on next page*

*Continued*

#### FlexPendant Command Server

By using the command line on the controller you can send commands to the FlexPendant. The FlexPendant has a command server that interprets the commands and performs the requested operation. The syntax for command line commands is `fpcmd "<command>"`. The only command you need to remember is `fpcmd "-h"`, which is the *Help* command. It produces a printout of available FlexPendant commands in the controller console:

```
-> fpcmd "-h" value = 8 = 0x8-> [fp]: FlexPendantCmd: Help
```

The following table lists the command with the specific actions:

Command	Action
<code>fpcmd "-h"</code>	Help
<code>fpcmd "-a"</code>	Adapter show routine
<code>fpcmd "-m"</code>	Measure time in adapters
<code>fpcmd "-i"</code>	Display FlexPendant Information
<code>fpcmd "-f"</code>	Bring GTPU Services to front
<code>fpcmd "-x"</code>	Hide startup progress bar
<code>fpcmd "-s"</code>	Start application
<code>fpcmd "-d"</code>	Copy Debug file to controller
<code>fpcmd "-as"</code>	Print screen
<code>fpcmd "-rd"</code>	Robot Communication Runtime debug
<code>fpcmd "-restart"</code>	Restart Device
<code>fpcmd "-memshow"</code>	Available memory
<code>fpcmd "-module"</code>	Module information of a process
<code>fpcmd "-filePut"</code>	Upload a file to the FlexPendant
<code>fpcmd "-filleted"</code>	Download a file to the FlexPendant
<code>fpcmd "-diarist"</code>	List a directory



#### NOTE!

All commands support `-?` (for example `fpcmd "-memShow -?"`), which gives further information about the specific command.



#### TIP!

It is possible to monitor memory consumption in the robot controller console window:

1. Write `fpcmd_enable_console_output 3`.
2. Write `fpcmd "-memShow"`.

For more information, see [Discover memory leaks on page 169](#).



---

#### Trace and Debug

The `ABB.Robotics.Diagnostics` namespace provides trace and debug services. Its `Trace` and `Debug` classes are specifically designed for the FlexPendant environment.

The properties and methods in the `Trace` class are used to instrument release builds, which allows you to monitor the health of your application running in a real-life setting. Tracing can help you to isolate problems and fix them without disturbing a running system.

If you use methods in the `Debug` class to print debugging information and check your logic with assertions, you can make your code more robust without impacting the performance and code size of your shipping product. In Visual Studio, creating a debug build enables `Debug`. `Trace` and `Debug` give printout information during execution. Messages are displayed on the FlexPendant screen or in the robot controller console. The functionality is similar to that provided by the `.NetTrace` and `Debug` classes.

The `Assert` method checks for a condition and displays an assert message on the FlexPendant, including detailed information and a stack trace, if the condition is false. The message is also displayed in the controller console window if you enter the command `fpcmd_enable_console_output` first.



#### **NOTE!**

Add the `ABB.Robotics.Diagnostics` namespace to the `using` section at the top of your source code file.

## 7.3. Debugging the virtual FlexPendant

---

### Overview

When debugging your application it is often convenient to use the virtual environment, but it is almost as easy to attach the Visual Studio debugger to the real FlexPendant device. For information about how that is done see [Debugging the FlexPendant device on page 190](#).

This section describes how to start the Visual Studio debugger, attach a running Virtual FlexPendant to it, set up break points and step through the source code of a FlexPendant application.

### Debugging procedure

There are several ways of attaching a Visual Studio debugger to a running Virtual FlexPendant application. In this section one method is described in detail.

There is no way to start your FlexPendant application from inside the Visual Studio environment. You must start by deploying the application to the Virtual FlexPendant in RobotStudio. How to do this is described in [Hands on - Hello world on page 68](#). Then you start the Virtual FlexPendant and attach the Visual Studio debugger to it.



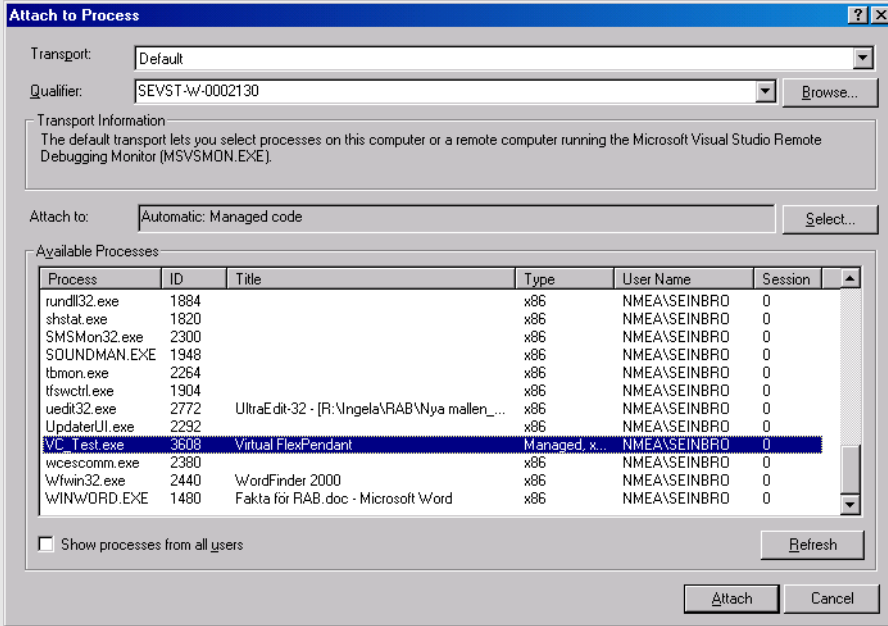
#### NOTE!

In order to use break points the project build configuration must be *Debug*. You set it in the the **Build** tab of the Project **Properties**. The output directory, where you find the assembly (\*.dll), the proxy assembly (\*gtpu.dll) and the program database (\*.pdb) is the bin/Debug directory, a sub-directory of your Visual Studio project.

Continued

## Attach to Process

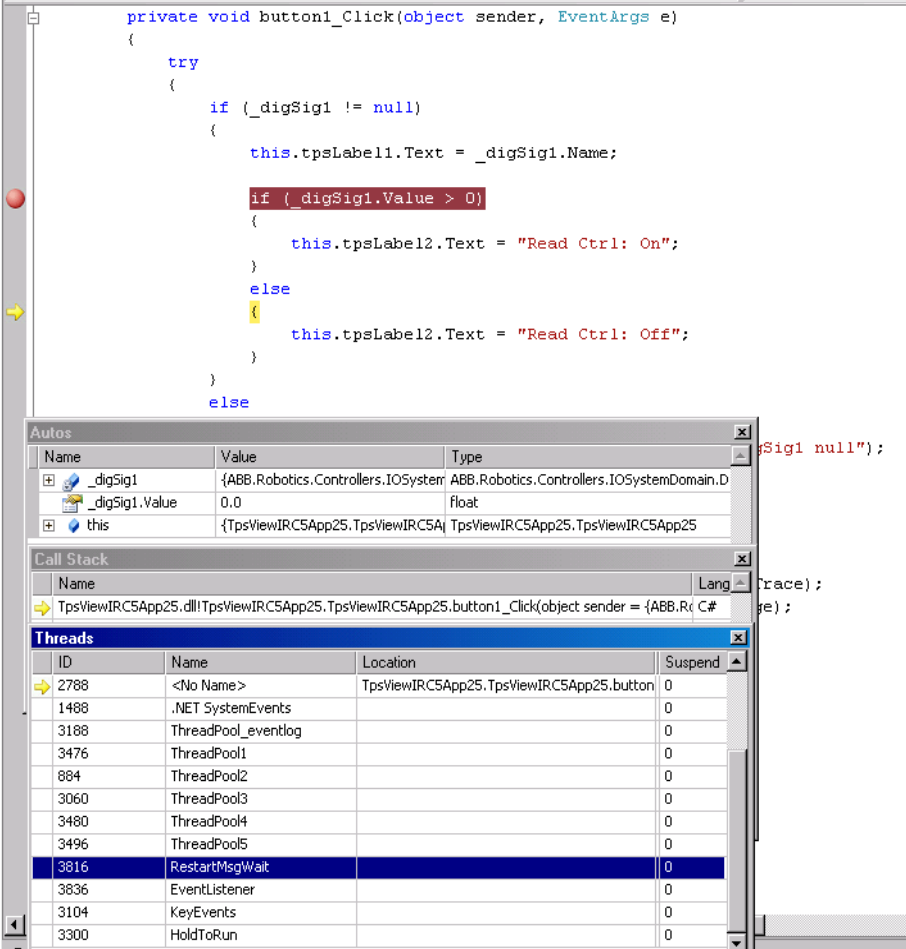
When you have a running application on the Virtual FlexPendant follow these steps:

Step	Action
1	<p>In Visual Studio on the <b>Debug</b> menu, select <b>Attach to Process</b>. It brings up this dialog:</p>  <p>9.1.3_1</p>
2	Select the <i>Virtual FlexPendant.exe</i> process and click the <b>Attach</b> button.
3	Set a break point in your source code.
4	On the Virtual FlexPendant, click a button of your application or something else that will make program execution hit the breakpoint.

## 7 FlexPendant - Debugging and troubleshooting

### 7.3. Debugging the virtual FlexPendant

Continued

Step	Action																																																																				
5	<p>On the <b>Debug</b> menu, point at Windows and select the debugging information to be displayed while debugging. See following example:</p>  <pre>private void button1_Click(object sender, EventArgs e) {     try     {         if (_digSig1 != null)         {             this.tpsLabel1.Text = _digSig1.Name;              if (_digSig1.Value &gt; 0)             {                 this.tpsLabel2.Text = "Read Ctrl: On";             }             else             {                 this.tpsLabel2.Text = "Read Ctrl: Off";             }         }     }     else     {         this.tpsLabel2.Text = "Read Ctrl: Off";     } }</pre> <table border="1"><caption>Autos</caption><thead><tr><th>Name</th><th>Value</th><th>Type</th></tr></thead><tbody><tr><td>_digSig1</td><td>{ABB.Robotics.Controllers.IOSystemDomain.D}</td><td>ABB.Robotics.Controllers.IOSystemDomain.D</td></tr><tr><td>_digSig1.Value</td><td>0.0</td><td>float</td></tr><tr><td>this</td><td>{TpsViewIRC5App25.TpsViewIRC5A}</td><td>TpsViewIRC5App25.TpsViewIRC5App25</td></tr></tbody></table> <table border="1"><caption>Call Stack</caption><thead><tr><th>Name</th><th>Lang</th></tr></thead><tbody><tr><td>TpsViewIRC5App25.dll:TpsViewIRC5App25.TpsViewIRC5App25.button1_Click(object sender = {ABB.R...</td><td>C#</td></tr></tbody></table> <table border="1"><caption>Threads</caption><thead><tr><th>ID</th><th>Name</th><th>Location</th><th>Suspend</th></tr></thead><tbody><tr><td>2788</td><td>&lt;No Name&gt;</td><td>TpsViewIRC5App25.TpsViewIRC5App25.button</td><td>0</td></tr><tr><td>1488</td><td>.NET SystemEvents</td><td></td><td>0</td></tr><tr><td>3188</td><td>ThreadPool_eventlog</td><td></td><td>0</td></tr><tr><td>3476</td><td>ThreadPool1</td><td></td><td>0</td></tr><tr><td>884</td><td>ThreadPool2</td><td></td><td>0</td></tr><tr><td>3060</td><td>ThreadPool3</td><td></td><td>0</td></tr><tr><td>3480</td><td>ThreadPool4</td><td></td><td>0</td></tr><tr><td>3496</td><td>ThreadPool5</td><td></td><td>0</td></tr><tr><td>3816</td><td>RestartMsgWait</td><td></td><td>0</td></tr><tr><td>3836</td><td>EventListener</td><td></td><td>0</td></tr><tr><td>3104</td><td>KeyEvents</td><td></td><td>0</td></tr><tr><td>3300</td><td>HoldToRun</td><td></td><td>0</td></tr></tbody></table>	Name	Value	Type	_digSig1	{ABB.Robotics.Controllers.IOSystemDomain.D}	ABB.Robotics.Controllers.IOSystemDomain.D	_digSig1.Value	0.0	float	this	{TpsViewIRC5App25.TpsViewIRC5A}	TpsViewIRC5App25.TpsViewIRC5App25	Name	Lang	TpsViewIRC5App25.dll:TpsViewIRC5App25.TpsViewIRC5App25.button1_Click(object sender = {ABB.R...	C#	ID	Name	Location	Suspend	2788	<No Name>	TpsViewIRC5App25.TpsViewIRC5App25.button	0	1488	.NET SystemEvents		0	3188	ThreadPool_eventlog		0	3476	ThreadPool1		0	884	ThreadPool2		0	3060	ThreadPool3		0	3480	ThreadPool4		0	3496	ThreadPool5		0	3816	RestartMsgWait		0	3836	EventListener		0	3104	KeyEvents		0	3300	HoldToRun		0
Name	Value	Type																																																																			
_digSig1	{ABB.Robotics.Controllers.IOSystemDomain.D}	ABB.Robotics.Controllers.IOSystemDomain.D																																																																			
_digSig1.Value	0.0	float																																																																			
this	{TpsViewIRC5App25.TpsViewIRC5A}	TpsViewIRC5App25.TpsViewIRC5App25																																																																			
Name	Lang																																																																				
TpsViewIRC5App25.dll:TpsViewIRC5App25.TpsViewIRC5App25.button1_Click(object sender = {ABB.R...	C#																																																																				
ID	Name	Location	Suspend																																																																		
2788	<No Name>	TpsViewIRC5App25.TpsViewIRC5App25.button	0																																																																		
1488	.NET SystemEvents		0																																																																		
3188	ThreadPool_eventlog		0																																																																		
3476	ThreadPool1		0																																																																		
884	ThreadPool2		0																																																																		
3060	ThreadPool3		0																																																																		
3480	ThreadPool4		0																																																																		
3496	ThreadPool5		0																																																																		
3816	RestartMsgWait		0																																																																		
3836	EventListener		0																																																																		
3104	KeyEvents		0																																																																		
3300	HoldToRun		0																																																																		
6	<p>On the <b>Debug</b> menu, select the appropriate <b>Step</b> command when stepping through your code.</p>																																																																				
7	<p>On the <b>Debug</b> menu, click <b>Detach All</b> or <b>Stop Debugging</b> when you want to stop debugging.</p>																																																																				

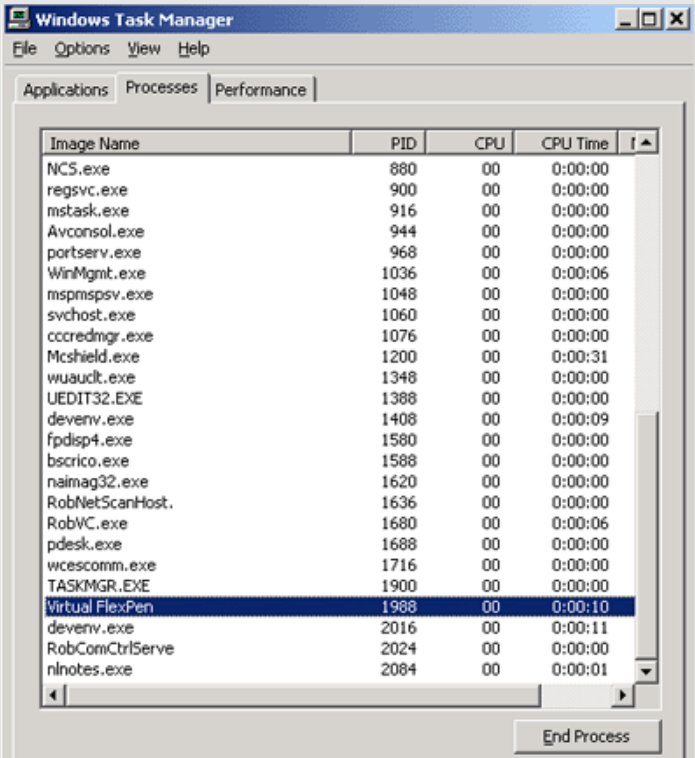
© Copyright 2010 ABB. All rights reserved.

Continues on next page

*Continued*

### Windows Task Manager

You can also attach a running application through the Windows Task Manager.

Step	Action																																																																																																								
1	<p>Start <i>Windows Task Manager</i> and select the <b>Processes</b> tab.</p>  <table border="1" data-bbox="603 600 1225 1153"> <thead> <tr> <th>Image Name</th> <th>PID</th> <th>CPU</th> <th>CPU Time</th> </tr> </thead> <tbody> <tr><td>NCS.exe</td><td>880</td><td>00</td><td>0:00:00</td></tr> <tr><td>regsvc.exe</td><td>900</td><td>00</td><td>0:00:00</td></tr> <tr><td>mstask.exe</td><td>916</td><td>00</td><td>0:00:00</td></tr> <tr><td>Avconsol.exe</td><td>944</td><td>00</td><td>0:00:00</td></tr> <tr><td>portserv.exe</td><td>968</td><td>00</td><td>0:00:00</td></tr> <tr><td>WinMgmt.exe</td><td>1036</td><td>00</td><td>0:00:06</td></tr> <tr><td>mspmbspv.exe</td><td>1048</td><td>00</td><td>0:00:00</td></tr> <tr><td>svchost.exe</td><td>1060</td><td>00</td><td>0:00:00</td></tr> <tr><td>ccredmgr.exe</td><td>1076</td><td>00</td><td>0:00:00</td></tr> <tr><td>Mcshield.exe</td><td>1200</td><td>00</td><td>0:00:31</td></tr> <tr><td>wuauclt.exe</td><td>1348</td><td>00</td><td>0:00:00</td></tr> <tr><td>UEDIT32.EXE</td><td>1388</td><td>00</td><td>0:00:00</td></tr> <tr><td>devenv.exe</td><td>1408</td><td>00</td><td>0:00:09</td></tr> <tr><td>fpdisp4.exe</td><td>1580</td><td>00</td><td>0:00:00</td></tr> <tr><td>bscrico.exe</td><td>1588</td><td>00</td><td>0:00:00</td></tr> <tr><td>naimag32.exe</td><td>1620</td><td>00</td><td>0:00:00</td></tr> <tr><td>RobNetScanHost.</td><td>1636</td><td>00</td><td>0:00:00</td></tr> <tr><td>RobVC.exe</td><td>1680</td><td>00</td><td>0:00:06</td></tr> <tr><td>pdesk.exe</td><td>1688</td><td>00</td><td>0:00:00</td></tr> <tr><td>wcescomm.exe</td><td>1716</td><td>00</td><td>0:00:00</td></tr> <tr><td>TASKMGR.EXE</td><td>1900</td><td>00</td><td>0:00:00</td></tr> <tr><td><b>Virtual FlexPen</b></td><td><b>1988</b></td><td><b>00</b></td><td><b>0:00:10</b></td></tr> <tr><td>devenv.exe</td><td>2016</td><td>00</td><td>0:00:11</td></tr> <tr><td>RobComCtrlServe</td><td>2024</td><td>00</td><td>0:00:00</td></tr> <tr><td>nlnotes.exe</td><td>2084</td><td>00</td><td>0:00:01</td></tr> </tbody> </table> <p>9.1.3_3</p>	Image Name	PID	CPU	CPU Time	NCS.exe	880	00	0:00:00	regsvc.exe	900	00	0:00:00	mstask.exe	916	00	0:00:00	Avconsol.exe	944	00	0:00:00	portserv.exe	968	00	0:00:00	WinMgmt.exe	1036	00	0:00:06	mspmbspv.exe	1048	00	0:00:00	svchost.exe	1060	00	0:00:00	ccredmgr.exe	1076	00	0:00:00	Mcshield.exe	1200	00	0:00:31	wuauclt.exe	1348	00	0:00:00	UEDIT32.EXE	1388	00	0:00:00	devenv.exe	1408	00	0:00:09	fpdisp4.exe	1580	00	0:00:00	bscrico.exe	1588	00	0:00:00	naimag32.exe	1620	00	0:00:00	RobNetScanHost.	1636	00	0:00:00	RobVC.exe	1680	00	0:00:06	pdesk.exe	1688	00	0:00:00	wcescomm.exe	1716	00	0:00:00	TASKMGR.EXE	1900	00	0:00:00	<b>Virtual FlexPen</b>	<b>1988</b>	<b>00</b>	<b>0:00:10</b>	devenv.exe	2016	00	0:00:11	RobComCtrlServe	2024	00	0:00:00	nlnotes.exe	2084	00	0:00:01
Image Name	PID	CPU	CPU Time																																																																																																						
NCS.exe	880	00	0:00:00																																																																																																						
regsvc.exe	900	00	0:00:00																																																																																																						
mstask.exe	916	00	0:00:00																																																																																																						
Avconsol.exe	944	00	0:00:00																																																																																																						
portserv.exe	968	00	0:00:00																																																																																																						
WinMgmt.exe	1036	00	0:00:06																																																																																																						
mspmbspv.exe	1048	00	0:00:00																																																																																																						
svchost.exe	1060	00	0:00:00																																																																																																						
ccredmgr.exe	1076	00	0:00:00																																																																																																						
Mcshield.exe	1200	00	0:00:31																																																																																																						
wuauclt.exe	1348	00	0:00:00																																																																																																						
UEDIT32.EXE	1388	00	0:00:00																																																																																																						
devenv.exe	1408	00	0:00:09																																																																																																						
fpdisp4.exe	1580	00	0:00:00																																																																																																						
bscrico.exe	1588	00	0:00:00																																																																																																						
naimag32.exe	1620	00	0:00:00																																																																																																						
RobNetScanHost.	1636	00	0:00:00																																																																																																						
RobVC.exe	1680	00	0:00:06																																																																																																						
pdesk.exe	1688	00	0:00:00																																																																																																						
wcescomm.exe	1716	00	0:00:00																																																																																																						
TASKMGR.EXE	1900	00	0:00:00																																																																																																						
<b>Virtual FlexPen</b>	<b>1988</b>	<b>00</b>	<b>0:00:10</b>																																																																																																						
devenv.exe	2016	00	0:00:11																																																																																																						
RobComCtrlServe	2024	00	0:00:00																																																																																																						
nlnotes.exe	2084	00	0:00:01																																																																																																						
2	Select the <i>Virtual FlexPendant.exe</i> process and right-click to get the context menu. In that menu select <b>Debug</b> . You will get a warning message, but select <b>Yes</b> .																																																																																																								
3	A <i>Just-In-Time Debugging</i> dialog will appear. Select your application project as the debugger to use.																																																																																																								

### Launching debugger programatically

Yet another way of attaching a debugger is to launch it programatically, by writing code in your application.

Step	Action
	Insert the following line where you want the debugger to start: <code>System.Diagnostics.Debugger.Launch()</code>
	Start the application in the Virtual FlexPendant and perform the action which is to launch the debugger. A <i>Just-In-Time Debugging</i> dialog will appear.
	Select your Visual Studio project as the debugger. Click <b>OK</b> and start the debug session.

## 7 FlexPendant - Debugging and troubleshooting

### 7.4. Debugging the FlexPendant device

#### 7.4. Debugging the FlexPendant device

##### Overview

This section provides information on how to do debug an application executing on the real FlexPendant device.

In general, using the Visual Studio debugger to debug the device works very well and is strongly recommended, but depending on the OS, Visual Studio version and FlexPendant version you are using the requirements for setting up and attaching the Visual Studio debugger will differ somewhat.

Further information and updates concerning this topic will from this release be published on the FlexPendant SDK User Forum (and not in the Application manual). For more information, see [FlexPendant SDK User Forum on page 192](#).

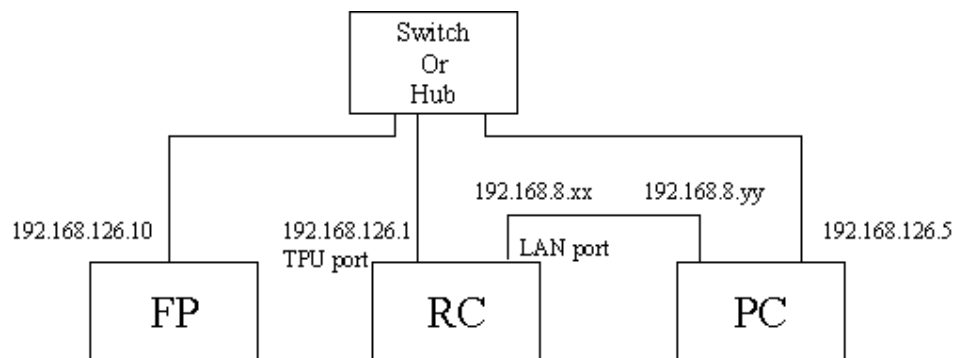
##### Prerequisites

The following requirements have to be met:

- Service Pack 1 or 2 for .NET Compact Framework 2.0 (.NET CF 2.0 SP2) is required for setting up and using the Visual Studio debugger on the FlexPendant device. It can be downloaded from <http://www.microsoft.com/downloads>.
- To debug with Visual Studio 2008 (as well as Visual Studio 2005 without SP1) you must follow a procedure that will be presented on the FlexPendant SDK User Forum. The procedure will be different for each RobotWare release.
- If your PC is running under Windows Vista “Windows Mobile Device Center” needs to be installed in order to connect to the device.

##### Setting up the network

This illustration shows how to connect the FlexPendant, the Robot Controller and your PC in order to debug your application using the Visual Studio debugger.



9.1.4\_3

The FlexPendant has a static IP address 192.168.126.10. Your PC IP address must be one on the 126 subnet, that is, 192.168.126.x (not 1 or 255).

Note that this setup is completely independent of the LAN and Service ports. You are plugging the TPU cable from the RC into the switch, a cable from the switch to the RC, and a cable from your PC to the switch.

A connection over the LAN port is optional, but useful, as you may need to use RSO or FTP at the same time. To use both connections, your PC requires two NICs.

*Continues on next page*

### Debugging procedure

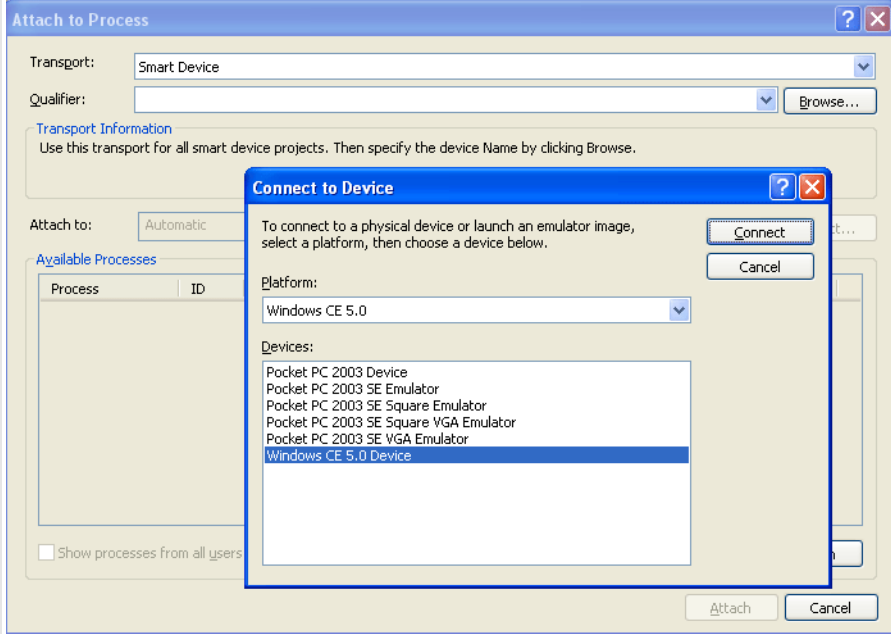
Follow these steps to set up and attach the Visual Studio 2005 SP1 debugger to the FlexPendant device:

Step	Action
1	On the <b>Tools</b> menu in Visual Studio 2005 click <b>Options</b> .
2	In the <b>Options</b> dialog expand the <b>Device Tools</b> node and select <b>Devices</b> . Select <b>Windows CE 5.0 Device</b> as shown in the following screenshot. <div data-bbox="555 526 1385 1012" data-label="Image"> </div> <p>9.1.4_6</p> <p><b>Note!</b> If your FlexPendant version is <i>SxTpu1</i> the option <i>Windows CE 5.0 Device</i> is not available. Instead you should select <i>Pocket PC 2003 Device</i>.</p>
3	When the right device has been selected click <b>Properties</b> and then <b>Configure</b> .
4	Apply the settings shown in the picture. <div data-bbox="555 1198 1104 1482" data-label="Image"> </div> <p>9.1.4_7</p>

## 7 FlexPendant - Debugging and troubleshooting

### 7.4. Debugging the FlexPendant device

Continued

Step	Action
5	On the <b>Debug/Tools</b> menu in Visual Studio click <b>Attach To Process</b> . In the dialog select <i>Smart Device</i> for <b>Transport</b> and click the <b>Browse</b> button to specify platform and device. Then click <b>Connect</b> .
	
	9.1.4_8
6	In the <b>Available Processes</b> list select <i>taf.exe</i> . Click <b>Attach</b> .
7	Set a break point in your source code.
8	On the FlexPendant, click a button of your application or something else that will make program execution hit the breakpoint.



#### NOTE!

If your PC is NOT using XP with Visual Studio 2005 SP1 you need to find information on how to attach the Visual Studio debugger for your specific environment on the FlexPendant SDK User Forum.

#### FlexPendant SDK User Forum

The *User Forum* of ABB's *RobotStudio Community* has a section dedicated to FlexPendant SDK. Here beginners as well as experts discuss code and solutions online. This is also where you find FlexPendant SDK releases for free download and any information that the development or support team want to share with you. For more information, see [RobotStudio Community on page 16](#).



## 7.5. Troubleshooting FlexPendant applications

### Overview

If you encounter problems when running your application there are a few things you should do before contacting your service organization. The following steps represent a rough guideline:

	Action
1	Is it possible to start your application? If not see <a href="#">FlexPendant application does not start on page 194</a> .
2	Have you checked the FlexPendant SDK Release Notes? Many questions will find an answer in the Release Notes of the specific release. These are available on the RobotWare DVD and at the Software Download Site.
3	Have you tried to pinpoint the problem by debugging the FlexPendant? If not see <a href="#">Debugging the FlexPendant device on page 190</a> for information about how to do it.
4	Have you tried to get debug printouts? For more information, see <a href="#">Debug output on page 183</a> .
5	Is the problem FlexPendant hangings? Make sure you use <code>Invoke</code> when modifying the user interface due to a robot controller event. For more information, see <a href="#">GUI and controller event threads in conflict on page 50</a> and <a href="#">Invoke method on page 51</a> . When a hanging occurs attach the Visual Studio debugger to the FlexPendant. On the <b>Debug</b> menu, point at <b>Windows</b> and select <b>Threads</b> . Examine the threads to discover any deadlocks.



### TIP!

If you still have not found a solution to your problem, take a look at the *User Forum* of *RobotStudio Community*, which includes a forum dedicated to discussion and chat on FlexPendant SDK, RAPID and IRC5 development topics. For more information, see [RobotStudio Community on page 16](#)

## 7 FlexPendant - Debugging and troubleshooting

### 7.5. Troubleshooting FlexPendant applications

Continued

#### FlexPendant application does not start

If you are unable to start your application the following table suggests possible scenarios.

Problem	Possible solution
The proxy assembly (*.gtpu.dll) is not built.	Correct any parameter error in the TpsView attribute. For more information, see <a href="#">FlexPendant TpsView attribute on page 39</a> .
The ABB Compliance Tool complains it cannot find the C# compiler.	It happens that the installation of Visual Studio does not set the path to the C# compiler properly. The C# compiler is necessary for running the ABB Compliance Tool. Confirm that the C# compiler is available by starting a command window and run "CSC.EXE". If the result is similar to this the path is properly set: C:\>csc.exe Microsoft (R) Visual C# .NET Compiler version 7.10.3052.4 for Microsoft (R) .NET Framework version 1.1.4322Copyright (C) Microsoft Corporation 2001-2002. All rights reserved. fatal error CS2008: No inputs specified If not, find CSC.EXE and copy its path to the <b>Properties</b> dialog of your Visual Studio project. Also add the path to the system PATH environment variables. For Windows 2000 Pro and Windows XP find the dialog for this at: Control Panel -> System -> Advanced -> Environment Variables -> System variables -> Add new. Add the path to the directory where the C# compiler is kept. Notice that a semicolon separates the path items. Verify in the command window afterwards that the CSC.EXE runs.
The application does not appear in the ABB menu.	Make sure the robot system has the <i>FlexPendant Interface</i> option.
Null reference exception or "Can't find object" when tapping the application icon in the ABB menu.	Make sure all arguments in the TpsView attribute are appropriate. For more information, see <a href="#">FlexPendant TpsView attribute on page 39</a> .
The bitmap constructor fails when the real FlexPendant tries to load your images, in the virtual environment this does not happen.	Change your images if they use more than 256 colors. The operating system of the first generation FlexPendant device (SxTPU1) only supports 256 colors.
The Virtual FlexPendant process remains alive.	If you forget to dispose an object that has a COM reference, such as the Controller object or a Signal, the Virtual FlexPendant process might stay alive. Go through the application and make sure that you dispose of all objects that have a Dispose method.

© Copyright 2010 ABB. All rights reserved.

Continues on next page

---

#### Important support information

If you cannot solve the problem on your own, make sure that before taking contact with a support organization this information is available:

- Written description of the problem.
- Application source code.
- System error logs.
- A backup of the system.
- Description of work-around if such exists.



#### **TIP!**

Even better than the complete application is a small repro program, which exposes your problem.

## 7 FlexPendant - Debugging and troubleshooting

---

### 7.5. Troubleshooting FlexPendant applications

## 8 Localizing a FlexPendant application

### 8.1. Adding support for several languages

#### Introduction

This chapter provides the information needed to localize a customized application. The FlexPendant has built-in support for localization, and the mechanisms used by the standard FlexPendant applications can also be used by FlexPendant SDK applications.

This enables customized applications to be presented in the active language of the FlexPendant, that is, the language selected in the standard *view Control Panel - Language*.

For this to work the texts displayed in the customized application must be translated and the application localized as described in this chapter.

#### Get started

Develop the application using English as the default language. The recommendation is to design, implement and test the application before adding support for other languages. To localize a FlexPendant application carefully complete each procedure of this chapter.


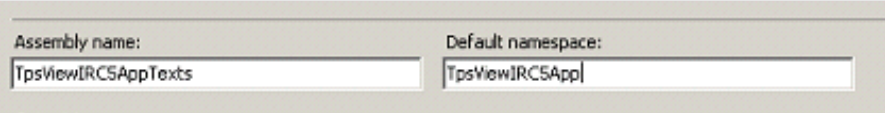


#### NOTE!

A method is needed to handle localization when new application functionality is added.

#### 1 Create project for text resources

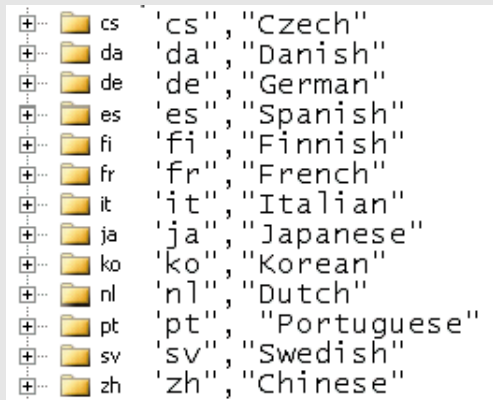



This procedure sets up a separate project for the user interface texts.

Step	Action
1.	<p>Create a new project in the solution. Choose a <i>Smart Device - Windows CE 5.0 - Empty Project</i> and name the project <i>&lt;YourAppName&gt;Texts</i>.</p>  <p><b>NOTE!</b> Both projects should belong to the same solution. The application will now compile to three assemblies: <i>&lt;YourAppName&gt;.dll</i> , <i>&lt;YourAppName&gt;.gtpu.dll</i> and <i>&lt;YourAppName&gt;Texts.dll</i>.</p>
2.	In the <i>Texts</i> project add a reference to <i>System (.Net)</i> .
3.	Set the <b>Output type</b> to <i>Class Library</i> (this is done in the <b>Project Properties</b> ).
4.	<p>The namespace used for the <i>Texts</i> project must be the same as used for the main project. As the namespace is not visible in the resource file you must change it in the <b>Project Properties</b> like this:</p> 

## 8 Localizing a FlexPendant application

### 8.1. Adding support for several languages

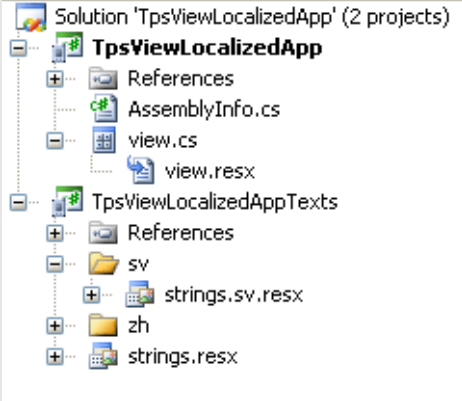

Continued

Step	Action																																																												
5.	Add a <i>Resources</i> file to the project by right clicking the project in <b>Solution Explorer</b> and selecting <b>Add New Item</b> . Set the name of the file to " <i>strings.resx</i> ". This file will contain the texts of the default language (normally English).																																																												
6.	Open the resource file and add <i>name</i> and <i>value</i> for the texts of the default language. Use capital letters in the <i>name</i> column.																																																												
	<table border="1"> <thead> <tr> <th colspan="6">Data for data</th> </tr> <tr> <th></th> <th>name</th> <th>value</th> <th>comment</th> <th>type</th> <th>mimetype</th> </tr> </thead> <tbody> <tr> <td>▶</td> <td>TXT_ABB_MENU_TITLE</td> <td>Your Application</td> <td>(null)</td> <td>(null)</td> <td>(null)</td> </tr> <tr> <td></td> <td>TXT_MESSAGE_BTN</td> <td>Message</td> <td>(null)</td> <td>(null)</td> <td>(null)</td> </tr> <tr> <td></td> <td>TXT_BOX_MSG</td> <td>Press one of the button</td> <td>(null)</td> <td>(null)</td> <td>(null)</td> </tr> <tr> <td></td> <td>TXT_BOX_CAPTION</td> <td>Message Caption</td> <td>(null)</td> <td>(null)</td> <td>(null)</td> </tr> <tr> <td></td> <td>TXT_RESPONSE_YES</td> <td>You pressed Yes</td> <td>(null)</td> <td>(null)</td> <td>(null)</td> </tr> <tr> <td></td> <td>TXT_RESPONSE_NO</td> <td>You pressed No</td> <td>(null)</td> <td>(null)</td> <td>(null)</td> </tr> <tr> <td></td> <td>TXT_RESPONSE_CANCEL</td> <td>You pressed Cancel</td> <td>(null)</td> <td>(null)</td> <td>(null)</td> </tr> <tr> <td>*</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> <p>6.5.1_1</p>	Data for data							name	value	comment	type	mimetype	▶	TXT_ABB_MENU_TITLE	Your Application	(null)	(null)	(null)		TXT_MESSAGE_BTN	Message	(null)	(null)	(null)		TXT_BOX_MSG	Press one of the button	(null)	(null)	(null)		TXT_BOX_CAPTION	Message Caption	(null)	(null)	(null)		TXT_RESPONSE_YES	You pressed Yes	(null)	(null)	(null)		TXT_RESPONSE_NO	You pressed No	(null)	(null)	(null)		TXT_RESPONSE_CANCEL	You pressed Cancel	(null)	(null)	(null)	*					
Data for data																																																													
	name	value	comment	type	mimetype																																																								
▶	TXT_ABB_MENU_TITLE	Your Application	(null)	(null)	(null)																																																								
	TXT_MESSAGE_BTN	Message	(null)	(null)	(null)																																																								
	TXT_BOX_MSG	Press one of the button	(null)	(null)	(null)																																																								
	TXT_BOX_CAPTION	Message Caption	(null)	(null)	(null)																																																								
	TXT_RESPONSE_YES	You pressed Yes	(null)	(null)	(null)																																																								
	TXT_RESPONSE_NO	You pressed No	(null)	(null)	(null)																																																								
	TXT_RESPONSE_CANCEL	You pressed Cancel	(null)	(null)	(null)																																																								
*																																																													
7.	In the <i>Texts</i> project create a folder with the culture short form as name, for example <i>de</i> for German and <i>sv</i> for Swedish.																																																												
	 <p>10.1_6</p>																																																												
	<p> <b>NOTE!</b> Russian (ru) has been added in RobotWare/FlexPendant SDK 5.11.</p> <p> <b>CAUTION!</b> The standard short forms listed in the preceding screenshot must be used.</p> <p> <b>NOTE!</b> To use Chinese or another language with non-western characters, you must use the FlexPendant font, <code>TpsFont</code>, for any UI controls. It internally checks what language is currently active on the FlexPendant and uses the correct font for that language.</p>																																																												

© Copyright 2010 ABB. All rights reserved.

Continues on next page

*Continued*

Step	Action
8.	Copy the “strings.resx” file to the folder(s) created in the previous step.
9.	<p>The name of the file used for the foreign resources should be <i>strings.&lt;culture&gt;.resx</i>, for example “<i>strings.sv.resx</i>” as in the following screenshot. Right click the file and rename it.</p>  <p>10.1_4</p>
10.	<p>Open the resource file and translate the texts in the <i>value</i> column. The <i>name</i> is the identity of the text and should remain the same.</p>  <p><b>NOTE!</b></p> <p>Obviously, texts might get longer or shorter when translated. You may therefore need to increase the sizes of some GUI controls when you have tested the application.</p>


## 8 Localizing a FlexPendant application

### 8.1. Adding support for several languages

Continued

#### 2 Prepare main project for localization

Follow these steps to add localization to your main project:

Step	Action
1.	Add a reference to <i>ABB.Robotics.Taf.Base</i> in the main project.
2.	<p>In the <code>TpsView</code> attribute of the view class insert the name of the Texts dll like this:</p> <pre>[assembly: TpsView("ABB_MENU_TITLE_TXT", "tpu-Operator32.gif", "tpu-Operator16.gif", "TpsViewIRC5App.dll", "TpsViewIRC5App.TpsViewIRC5App", StartPanelLocation.Left, TpsViewType.Static, "TpsViewLocalizedAppTexts.dll", TpsViewStartupTypes.Manual)]</pre>
3.	<p>Declare a <code>TpsResourceManager</code> object at the top of the class as a private member variable and initialize it in the constructor.</p> <p>Add a call to an <code>InitializeTexts</code> method.</p> <pre>//declaration private ABB.Robotics.Tps.Resources.TpsResourceManager     _tpsRM;  //constructor method _tpsRM = new     ABB.Robotics.Tps.Resources.TpsResourceManager("TpsViewLocalizedApp.strings",     ABB.Robotics.Taf.Base.TafAssembly.Load("TpsViewLocalizedAppTexts.dll"));  InitializeComponent(); InitializeTexts();</pre> <p> <b>NOTE!</b></p> <p>The first constructor argument should be the name of your application with <i>.strings</i> as an added suffix. The second argument is the name of the assembly containing the resources.</p>
4.	<p>Implement <code>InitializeTexts()</code>. Use the <code>TpsResourceManager</code> object and call <code>GetString()</code> using the identity (<i>name</i>) of the text you want as argument. Depending on the active language of the FlexPendant this call will retrieve the corresponding language resource.</p> <p>Example:</p> <pre>InitializeTexts() {     this.Label1.Text = _tpsRM.GetString("TXT_INSTR_LABEL"); }</pre> <p>Leave the contents of the <code>InitializeComponent</code> method as it is, for example <code>Label1.Text = "Signals"</code> and so on.</p>

© Copyright 2010 ABB. All rights reserved.

Continues on next page

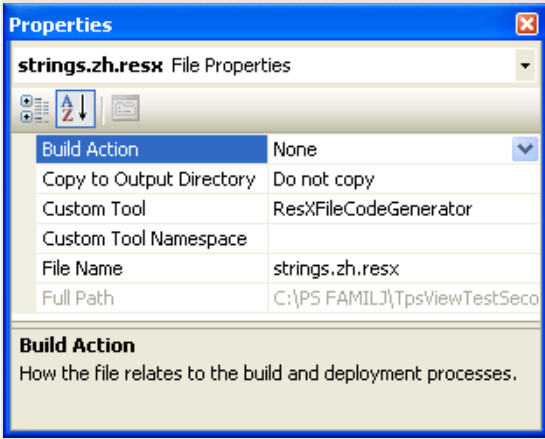



*Continued*

Step	Action
5.	<p>The <code>TpsResourceManager</code> object has not yet been created when the application icon and title are to appear in the ABB menu, and therefore another technique must be used to have them correctly displayed. Add a resource <i>name</i> for the application title and a <i>value</i> in the resource file of each language.</p> <p>In the <code>TpsView</code> attribute the first argument is the application title. Replace it with the resource <i>name</i>. It may look like this:</p> <pre>[assembly: TpsView("ABB_MENU_TITLE_TXT", . . . . . TpsViewIRC5AppTexts.dll ) ]</pre> <p>The corresponding resource <i>value</i> will now be used. In case no resource is found the <i>name</i> will be used as is.</p>

### 3 Build satellite assembly

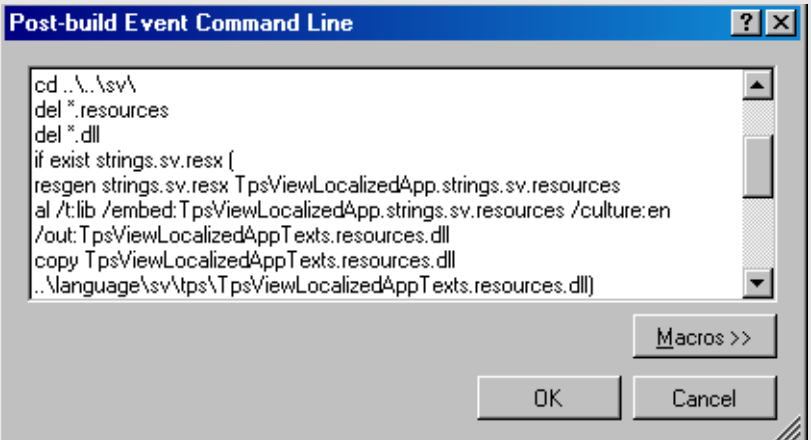

Follow these steps to create a satellite assembly of the localized resx file:

Step	Action
1.	<p>The localized resource file should not be built with the ordinary build process, that is, the property <b>Build Action</b> should be set to "None". Right click on the strings.&lt;culture&gt;.resx file and select properties:</p>  <p style="font-size: small;">6.5.1_4</p>
2.	<p>You should now use the Visual Studio 2005 tool <code>resgen.exe</code> to compile the resx file to a binary resource. After this the Visual Studio 2005 assembly linker tool <code>al.exe</code> should make a satellite assembly of the binary resource.</p> <p>Read the following steps very carefully to make sure the satellite assembly is built correctly.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">  <p><b>NOTE!</b> Localization with Visual Studio 2008 has not yet been tested.</p> </div>

## 8 Localizing a FlexPendant application

### 8.1. Adding support for several languages

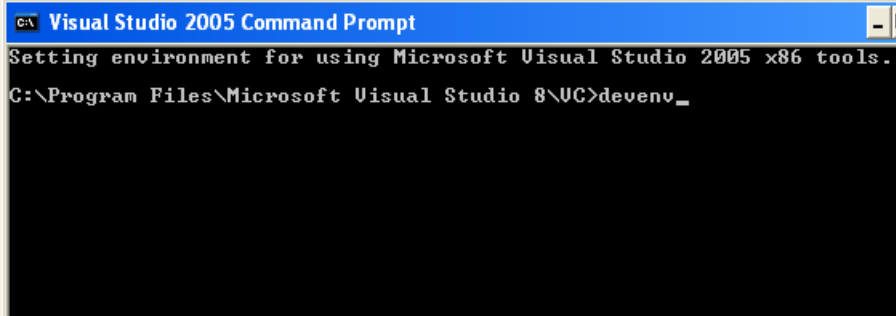

Continued

Step	Action
3.	<p>Create a post-build event in the <i>Texts</i> project in order to automate the building process.</p> <p>Example: <i>TpsViewLocalizedApp</i> with resources in Swedish:</p> <pre>mkdir ..\..\language mkdir ..\..\language\sv mkdir ..\..\language\sv\tps cd ..\..\sv\ del *.resources del *.dll if exist strings.sv.resx ( resgen strings.sv.resx TpsViewLocalizedApp.strings.sv .resources al /t:lib /embed:TpsViewLocalizedApp.strings.sv.resources /culture:en / out:TpsViewLocalizedAppTexts.resources.dll copy TpsViewLocalizedAppTexts.resources.dll ..\language\sv\tps\TpsViewLocalizedAppTexts.resource s.dll)</pre>  <pre>10.1_9</pre> <p>The <code>resgen</code> command is written like this:</p> <pre>resgen strings.&lt;culture&gt;.resx &lt;Namespace&gt;.strings.&lt;culture&gt;.resources</pre> <p>where <code>&lt;culture&gt;</code> should be replaced with the correct language short form and <code>&lt;Namespace&gt;</code> should be replaced with the application namespace.</p> <p>The <code>al</code> command takes the resulting dll located in the same directory as the <code>resx</code> file and makes a satellite assembly of it:</p> <pre>al /t:lib /embed:&lt;Namespace&gt;.strings.&lt;culture&gt;.resources / culture:en /out:&lt;AssemblyName&gt;.resources.dll</pre> <p> <b>NOTE!</b></p> <p>The name of the satellite assembly will be the same for all localized languages. The third argument of the <code>al</code> command, <code>culture:en</code>, should be “en”. The reason is that the FlexPendant operating system has English as the underlying language.</p>

© Copyright 2010 ABB. All rights reserved.

Continues on next page

*Continued*

Step	Action
4.	<p>It is necessary to ensure that the post-build step is executed with the correct versions of <i>resgen.exe</i> and <i>al.exe</i>. The easiest way to do this is to use a Visual Studio 2005 Command prompt to build the project (or solution) or to start Visual Studio from that prompt.</p> <p>First click Windows Start menu to launch the Command Prompt (at Programs &gt; Microsoft Visual Studio 2005 &gt; Visual Studio Tools). Then use the command <code>devenv</code> in order to start Visual Studio 2005.</p> <p>Now open your solution and build it. The post-build command is now guaranteed to execute with the correct settings.</p>  <p>6.5.1_5</p>  <p><b>NOTE!</b></p> <p>As default, Visual Studio will run post-build commands using the PC's user settings (paths and so on.). If you had Visual Studio 2003 installed earlier the post-build command is therefore very likely to use the wrong versions of <i>resgen.exe</i> and <i>al.exe</i>. The preceding procedure described guarantees that the Visual Studio 2005 versions are used.</p> <p>(These can be found at: C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\bin/resgen.exe C:\WINNT\Microsoft.NET\Framework\v2.0.50727/al.exe)</p>

## 8 Localizing a FlexPendant application

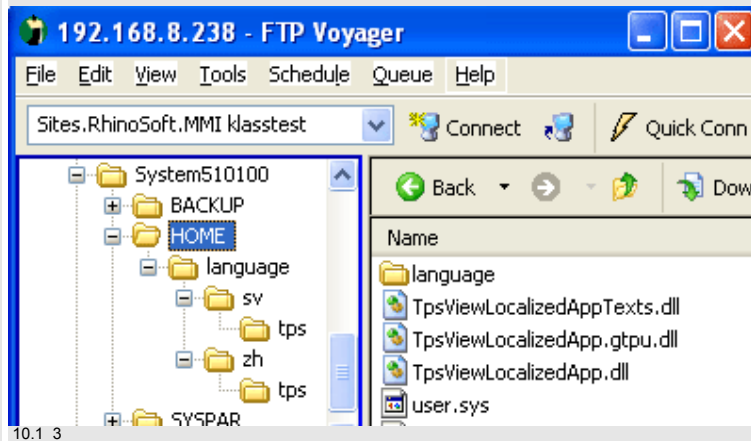
### 8.1. Adding support for several languages

Continued

#### 4 Test a localized application

In order to test a localized application, the resources must be correctly organized in the file system of the controller. This should be done in the same way either the test is done on a virtual or a real FlexPendant. Follow these steps to test the application on a real FlexPendant:

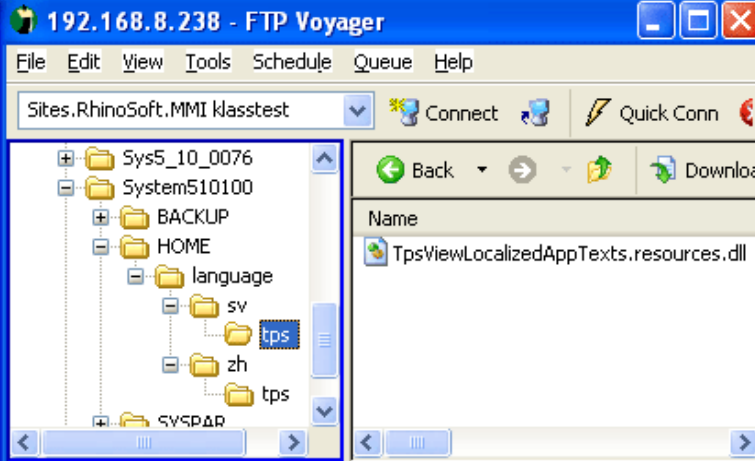

Step	Action
1.	Transfer, using an FTP client or the <i>File Manager</i> of RobotStudio, your application assemblies to the HOME directory of the active system. ( <i>TpsViewLocalizedApp-Texts.dll</i> , <i>TpsViewLocalizedApp.gtpu.dll</i> , <i>TpsViewLocalizedApp.dll</i> in the following screenshot).
2.	Copy the <i>language</i> folder that was created by the post-build event to the HOME directory.



#### NOTE!

It is also possible to have both the assemblies and the language folder located under the SYSTEM folder one level up. The advantage of using HOME is that the files are included in a backup.

Continued

Step	Action
3.	<p>Verify that the language folder has folders for each language and that each <i>tps</i> folder have a <i>.resources.dll</i>. If this is not the case, you need to go back to the post-build event of your <i>Texts</i> project and check what has gone wrong.</p>
	 <p>The screenshot shows the FTP Voyager interface connected to '192.168.8.238'. The directory tree on the left shows a path: System510100 &gt; language &gt; sv &gt; tps. The right pane displays the file 'TpsViewLocalizedAppTexts.resources.dll'.</p>
10.1_3b	
	
	<p><b>NOTE!</b></p>
	<p>The name of the satellite assembly is the same for all languages, <i>TpsViewLocalizedAppTexts.resources.dll</i> in the preceding example.</p>
4.	<p>Switch to another language (Control Panel - Languages) and restart the FlexPendant.</p>
5.	<p>Verify that the name of your application in the ABB menu is correct.</p>
6.	<p>Open the application and verify that the GUI texts are correctly displayed. Especially verify that text boxes, labels, listview columns and so on. are wide enough to display the translated texts.</p>

## 8 Localizing a FlexPendant application

---

### 8.1. Adding support for several languages

# 9 Deployment of a FlexPendant SDK application

## 9.1. Overview

### Introduction

For the end customer to use your application, it has to be deployed to the customer's robot controller. This is done when the customer robot system is created, by using the *System Builder* in RobotStudio. The custom application can then either be added as an additional option by using a license key, or added to the *Home* directory of the controller file system by using one of the dialog box of the *System Builder* wizard.

For an application with multi-language support there are a few more things to deal with.



#### NOTE!

Using an FTP client to upload the application from a PC to a robot controller can be done for testing purposes. It can also be done if the custom application needs to be added to an existing system, which is already running in production. For more information, see [Deployment using FTP on page 214](#).

### Making a product

These are the steps to make a product of a custom FlexPendant application:

1. Approval of the FlexPendant SDK product requirement specification.
2. Design and development of a FlexPendant SDK GUI prototype.
3. Approval of the FlexPendant SDK GUI prototype.
4. Design and development of a FlexPendant SDK functional prototype.
5. Approval of the FlexPendant SDK functional prototype.
6. Design and development of a FlexPendant SDK product.
7. Approval of the FlexPendant SDK product.
8. Design and development of a deployable FlexPendant SDK product.

### Deployment of a FlexPendant SDK product

Before deploying a custom application you need to consider these issues:

- Should the product be licensed, that is, be sold as an option?
- Is there a need to localize the product, that is, create support for native languages?

Depending on how the preceding questions are answered there are four alternatives (detailed separately in the following sections of this manual):

1. License and localization
2. License but no localization
3. No license but localization
4. No license and no localization



#### NOTE!

If the product is to be licensed it should be deployed as an additional option. If not, RobotStudio should be used to deploy the application to the *Home* directory of the system.

## 9 Deployment of a FlexPendant SDK application

### 9.2. Deployment of an application without a license

#### 9.2. Deployment of an application without a license

##### Overview

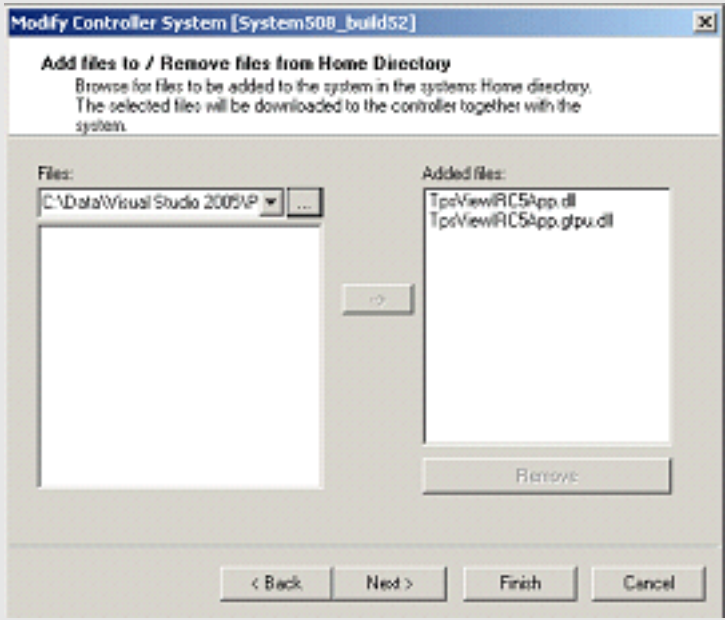
If you do not make an additional option of your application, the end user does not need a license to use it.

When the customer system is created, by using the *System Builder* of RobotStudio, your application should be added to the *Home* directory of the system.

This section gives information about how this is done. The easiest alternative, which offers no support for additional languages, is explained first.

##### No license and no localization

Use the following procedure to deploy an application without license nor multi-language support.

Step	Action
1.	Use <i>System Builder</i> in RobotStudio.
2.	Add the application assemblies(.dll) and other resources (.jpg, *.gif,*.bmp) to the <i>Home</i> directory: 
3.	Download the system to the robot controller.



##### NOTE!

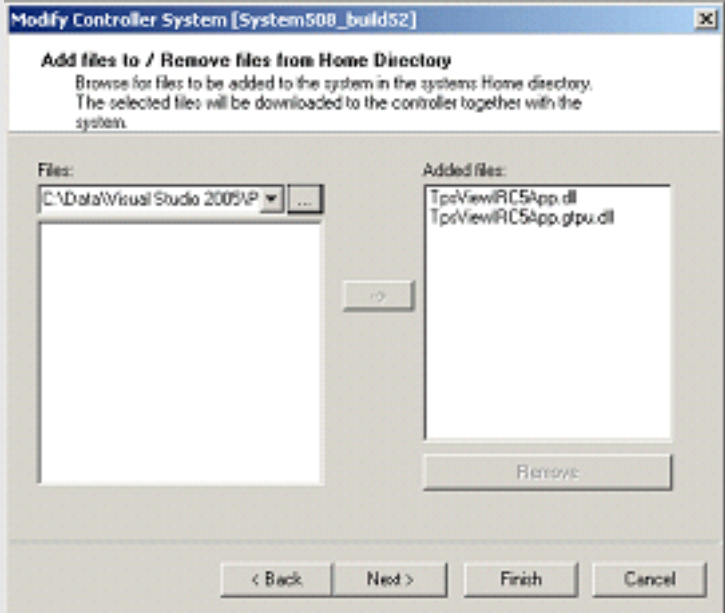
Having the application deployed to the *Home* directory means it will be included in a system backup.



*Continued*

#### No license but localization

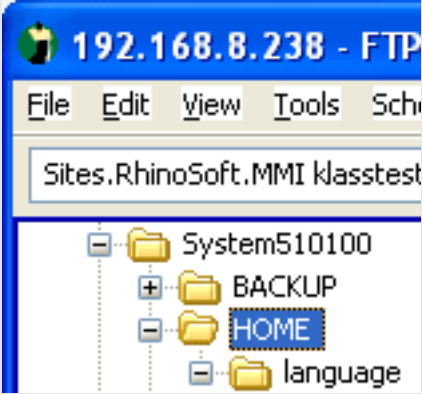

Use the following procedure to deploy an application with no license but with multi-language support.

Step	Action
1.	Implement multi-language support. For more information, see <a href="#">Localizing a FlexPendant application on page 197</a> .
2.	Use <i>System Builder</i> in RobotStudio.
3.	Add the application assemblies(.dlls) and other resources (*.jpg, *.gif,*.bmp) to the <i>Home</i> directory: 
	10.1_2 <b>Note!</b> <i>TpsViewIRC5AppTexts.dll</i> is missing in the figure.
4.	Generate the robot system.

## 9 Deployment of a FlexPendant SDK application

### 9.2. Deployment of an application without a license

Continued

Step	Action
5.	<p>In the generated RobotWare system, add a <i>language</i> directory with all supported languages in the <i>Home</i> directory. Then for each supported language, add a culture specific and a <i>tps</i> directory.</p>  <p>10.1_3</p>  <p><b>NOTE!</b></p> <p>Standard names (de, zh and so on.) must be used for the different languages. For complete list, see <a href="#">1 Create project for text resources on page 197</a>.</p>
6.	<p>Transfer each resource binary to the tps sub-directory of the respective culture directory, for example:</p> <p><i>TpsViewIRC5App.strings.de.resources.dll</i> to <i>language/de/tps</i> directory and so on.</p>
7.	<p>Download the system to the robot controller.</p> <p>For more information on how to add support for native languages to your custom application, see <a href="#">Localizing a FlexPendant application on page 197</a>.</p>

### 9.3. Deployment of a licensed application

#### Overview

When the customer system is created by *System Builder* of RobotStudio, a FlexPendant application can be added to the system as an additional option.

This section describes how to make the additional option, which is necessary for deployment of a licensed custom application. It also describes how the customer installs the option.



#### CAUTION!

An additional option must be distributed in accordance with RobotWare version and revision. Pay attention if you are using functionality, which has been included in a revision!

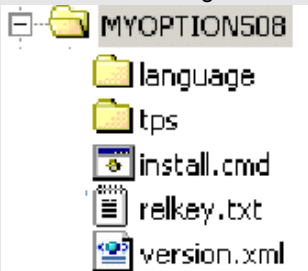



#### NOTE!

An additional option must be deployed with the structure of RobotStudio.

#### Procedure for making an additional option

Use the following procedure to make an additional option of a FlexPendant SDK application.

Step	Action
1.	Implement multi-language support if considered necessary. For more information, see <a href="#">Localizing a FlexPendant application on page 197</a> .
2.	Order a license and a CD Key Maker from your local ABB office, who will in turn contact ABB SEROP product support in Sweden.
3.	<p>Create the following structure:</p>  <p>10.1_12</p> <p>The <i>language</i> folder is needed if there is to be support for other languages.</p>  <p><b>NOTE!</b> Always use capital letters for the option name.</p>

## 9 Deployment of a FlexPendant SDK application

### 9.3. Deployment of a licensed application

Continued

Step	Action
4.	Create the <i>version.xml</i> file. It may look like this: <pre>&lt;Version&gt;   &lt;Major&gt;5&lt;/Major&gt;   &lt;Minor&gt;08&lt;/Minor&gt;   &lt;Revision&gt;00&lt;/Revision&gt;   &lt;Build&gt;0&lt;/Build&gt;   &lt;Title&gt;MYOPTION508&lt;/Title&gt;   &lt;Description&gt;My Optional FlexPendant Application&lt;/Description&gt;   &lt;Date&gt;2006-04-30&lt;/Date&gt;   &lt;Type&gt;AdditionalOption&lt;/Type&gt; &lt;/Version&gt;</pre> 10.1_13
5.	Create the <i>install.cmd</i> file. It may look like this: <pre>echo -text "Installing My Optional FlexPendant Application" register -type option -description MYOPTION508 -path \$BOOTPATH</pre> 10.1_14
6.	Create <i>relkey.txt</i> file. It may look like this: <pre>title MYOPTION508</pre> 10.1_15
7.	With the license, which is entered in the CD key Maker, generate a key file including the serial number of the Robot controller, for example MYOPTION508.kxt (This is the file you will enter in <i>System Builder</i> when the robot system is created.)
8.	Package the product on CD the way your organization recommends.



#### NOTE!

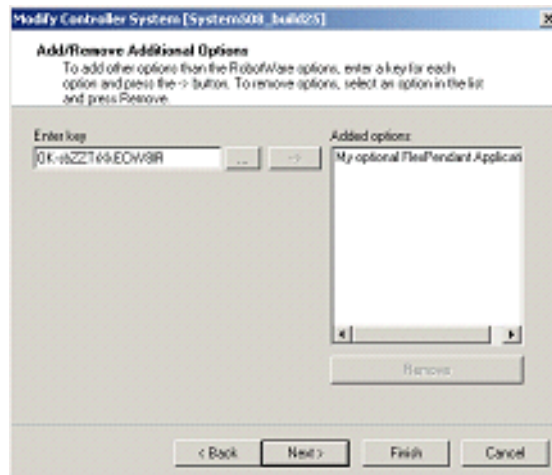
If the option is only a FlexPendant SDK application the described procedure is enough. But if the option should also include RAPID and configuration, you need to read about how to make an additional option in another manual, which is for ABB internal use. See *Related information* at the end of this section.

*Continued*

#### Installing the option at the customer

The following procedure shows how the customer installs a licensed application:

1. Install the additional option from a CD.
2. Create the robot system by using System Builder.
3. In the *Add Parameters/Add Additional Options* dialog browse to the key file, for example MYOPTION508.kxt which came with the installation of the additional option.



10.1\_16

#### Related information

Application manual - Additional Options, 3HAC023668-001

The manual is intended for ABB use only. Contact After Sales in Västerås, Sweden.

## 9 Deployment of a FlexPendant SDK application

---

### 9.4. Deployment using FTP

#### 9.4. Deployment using FTP

---

##### Overview

The general rule is that deployment using an FTP client should only be done during the development phase.

If deployment to a customer is done this way each controller has to be individually updated with assembly files as well as graphical and language resources. The organization of files in the robot controller file system is the responsibility of the application developer or the system integrator.

##### Procedure

Follow these steps to deploy your application using for example the *File Manager* of RobotStudio or an FTP client such as *Ftp Voyager*:

Step	Action
1	On the controller navigate to the system you want to update with the FlexPendant application.
2	Transfer the assembly, the proxy assembly and graphical resources used by the application to the system directory <i>Home</i> .
3	For multi-language support, in the <i>Language</i> directory create sub-directories for each language using the short name of the culture.
4	Create a <i>"tps"</i> sub-directory in each of these directories.
5	Copy each language resource to the <i>tps</i> folder of the corresponding culture.
6	Restart the FlexPendant. The custom application should now be accessible from the ABB menu. For more information on how to restart the FlexPendant but not the controller., see <a href="#">Restart the FlexPendant on page 35</a> .

**A**

- AlphaPad 94
  - Closing event 95
  - Launching 94
  - Property window 95
  - Removing 96
- Application 13
  - Architecture 31
  - assembly 40
  - class name 41
  - Deployment 28, 33
  - Development 25
  - Dll 32
  - icon 40
  - language 25
  - Life Cycle 37
  - location 41
  - name 40
  - Object oriented 48
  - Porting 28
  - Real 28
  - Running 32
  - TaskBar icon 40
  - TpsView 39
  - Type 41
  - UAS 52
  - Versions 21
  - Virtual 27

**C**

- C-API domains 116
- CloseView 112
- CompactAlphaPad 99
- compatibility 45
- Configuration 60
- console window 24
  - Commands 184

**D**

- DataBinding 105
  - AdvancedBinding 109
  - Classes 105
  - Definition 105
  - RapidDataBindingSource 105
  - ResumeBinding 110
  - SignalBindingSource 105
  - SuspendBinding 110
- dialog box 114
  - FpRapidData 114
  - FpToolCalibration 114
  - FpWorkObjectCalibration 114
- Dispose 121, 168

**F**

- FlexPendant
  - Applications 21
  - Communication 36
  - Memory 165
  - memory leaks 169
  - overview 165

- Platform 31
- Restarting 35
- Screen 165

**G**

- Grants 53
  - Definition 52
  - DemandGrants 53
  - GetCurrentGrants 53
- GTPUFolderBrowserDialog 102
- GTPUOpenFileDialog 102
- GTPUSaveFileDialog 102

**H**

- Hello world 68

**I**

- ImageList 88, 98
- Installation 19
  - Requirements 20
- Invoke 51
- ITpsViewActivation 171
- ITpsViewSetUp 111

**L**

- LaunchView 112
- ListView 97
  - Illustration 97
  - MultiSelect 98
  - Scrollable 98
  - SelectionEnabledOverScrollbuttons 98
  - ShowNumberOfItems 98
  - ShowSelectio 98

**M**

- Memory Leaks 77

**N**

- NumPad 99

**P**

- PictureBox 88
- Project 62
  - Empty 82
  - Form 82
  - New Project 62
  - Procedure 62
  - Wizard 72
- Properties window 104
  - FileName 104
  - Filter 104
  - InitialDirectory 104

**R**

- Rank 130
- RAPID 145
  - arrays 130
  - arrays dimensions 130
  - BeginInvoke 51
  - Event log 156
  - Invoke 51
  - Load Modules 147

- MessageBox 53
- ModifyPosition 149
- MotionPointer 148
- ProgramPointer 148
- RestProgramPointer 146
- Save and unload 148
- Save module 148
- Start 145
- Stop 145
- TpsControl 51

ReadItem 133

### **S**

- Signals 150
  - accessing 150
  - Listening changes 154
  - SignalChangedEventArgs 155
  - SignalFilter 151
  - values 152

### **T**

- TpsFont 87
- TpsForm 82
- Try-catch-finally 55
- Typecasting 55

### **U**

- User Forum 192

### **V**

- virtual keyboard 94

### **W**

- WriteItem 133