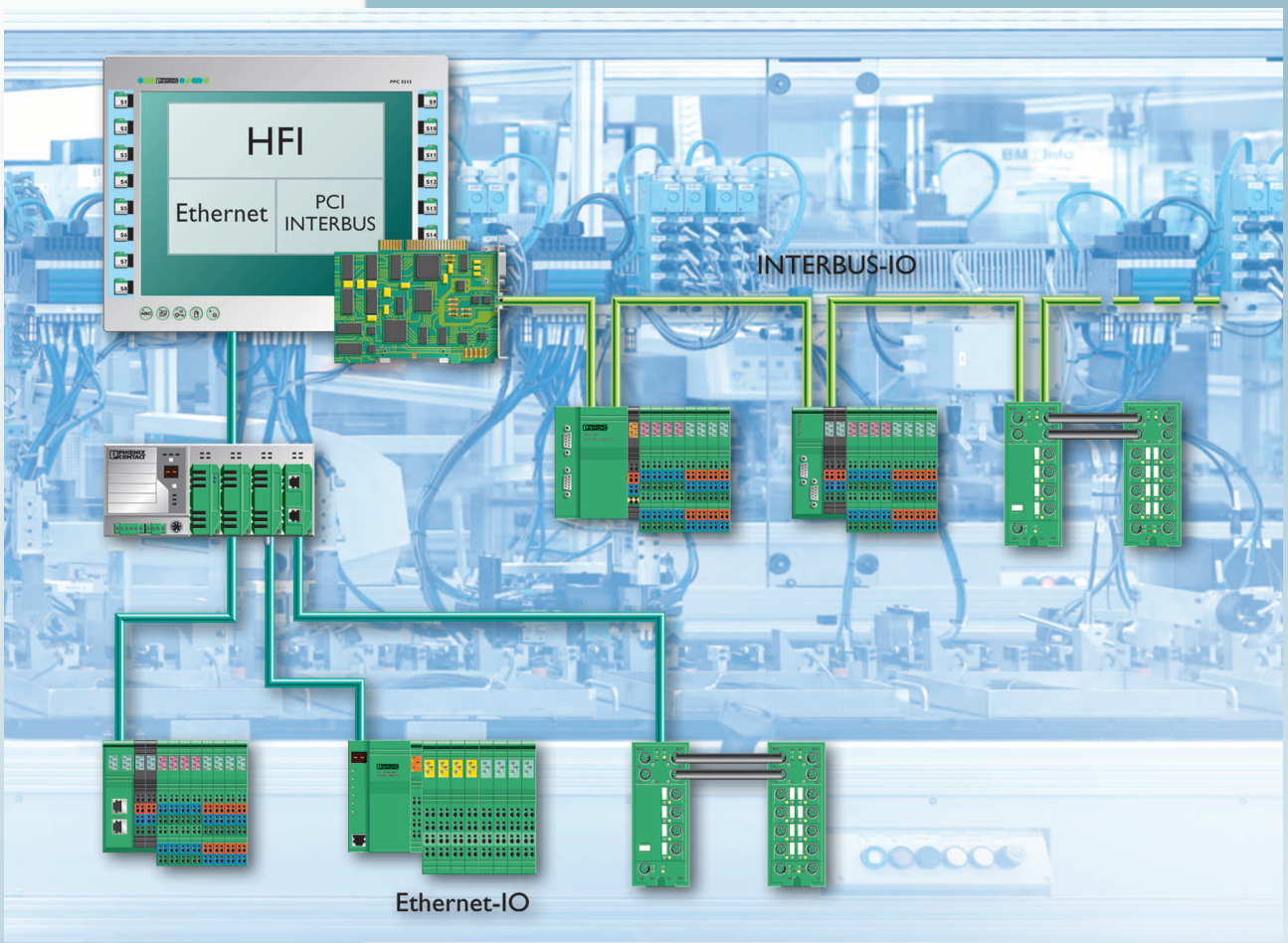
AUTOMATIONWORX



Quick Start Guide

UM QS EN HFI PROG

Order No.: 2910240

Programming in High-Level Language
Using the HFI User Interface

PHOENIX CONTACT
INSPIRING INNOVATIONS

AUTOMATIONWORX

Quick Start Guide

Programming in High-Level Language Using the HFI User Interface

05/2007

Designation: UM QS EN HFI PROG

Revision: 01

Order No.: 2910240

This user manual is valid for:

Designation	Version
HFI	1.1x
HFI	2.0x

Please Observe the Following Notes

In order to ensure the safe use of the product described, we recommend that you read this manual carefully. The following notes provide information on how to use this manual.

User Group of This Manual

The use of products described in this manual is oriented exclusively to qualified application programmers and software engineers, who are familiar with the safety concepts of automation technology and applicable standards.

Phoenix Contact accepts no liability for erroneous handling or damage to products from Phoenix Contact or third-party products resulting from disregard of information contained in this manual.

Explanation of Symbols Used

The *attention* symbol refers to an operating procedure which, if not carefully followed, could result in damage to hardware and software or personal injury.

The *note* symbol informs you of conditions that must be strictly observed to achieve error-free operation. It also gives you tips and advice on the efficient use of hardware and on software optimization to save you extra work.

The *text* symbol refers to detailed sources of information (manuals, data sheets, literature, etc.) on the subject matter, product, etc. This text also provides helpful information for the orientation in the manual.

We Are Interested in Your Opinion

We are constantly striving to improve the quality of our manuals.

Should you have any suggestions or recommendations for improvement of the contents and layout of our manuals, please send us your comments.

PHOENIX CONTACT GmbH & Co. KG
Documentation Services
32823 Blomberg
Germany

Phone +49 - 52 35 - 30 0
Fax + 49 - 52 35 - 34 20 21
E-mail tecdoc@phoenixcontact.com

General Terms and Conditions of Use for Technical Documentation

Phoenix Contact GmbH & Co. KG reserves the right to alter, correct, and/or improve the technical documentation and the products described in the technical documentation at its own discretion and without giving prior notice, insofar as this is reasonable for the user. The same applies to any technical changes that serve the purpose of technical progress.

The receipt of technical documentation (in particular data sheets, installation instructions, manuals, etc.) does not constitute any further duty on the part of
Phoenix Contact GmbH & Co. KG to furnish information on alterations to products and/or technical documentation. Any other agreement shall only apply if expressly confirmed in writing by Phoenix Contact GmbH & Co. KG.
Please note that the supplied documentation is product-specific documentation only and that you are responsible for checking the suitability and intended use of the products in your specific application, in particular with regard to observing the applicable standards and regulations.
Although Phoenix Contact GmbH & Co. KG makes every effort to ensure that the information content is accurate, up-to-date, and state-of-the-art, technical inaccuracies and/or printing errors in the information cannot be ruled out. Phoenix Contact GmbH & Co. KG does not offer any guarantees as to the reliability, accuracy or completeness of the information.
All information made available in the technical data is supplied without any accompanying guarantee, whether expressly mentioned, implied or tacitly assumed. This information does not include any guarantees regarding quality, does not describe any fair marketable quality, and does not make any claims as to quality guarantees or guarantees regarding the suitability for a special purpose.

Phoenix Contact GmbH & Co. KG accepts no liability or responsibility for errors or omissions in the content of the technical documentation (in particular data sheets, installation instructions, manuals, etc.).

The aforementioned limitations of liability and exemptions from liability do not apply, in so far as liability must be assumed, e.g., according to product liability law, in cases of premeditation, gross negligence, on account of loss of life, physical injury or damage to health or on account of the violation of important contractual obligations. Claims for damages for the violation of important contractual obligations are, however, limited to contract-typical, predictable damages, provided there is no premeditation or gross negligence, or that liability is assumed on account of loss of life, physical injury or damage to health. This ruling does not imply a change in the burden of proof to the detriment of the user.

Statement of Legal Authority

Internet

Up-to-date information on Phoenix Contact products can be found on the Internet at:

www.phoenixcontact.com

Make sure you always use the latest documentation.
It can be downloaded at:

www.download.phoenixcontact.com

A conversion table is available on the Internet at:

www.download.phoenixcontact.com/general/7000_en_00.pdf

Table of Contents

1 General

1.1 Purpose of This Quick Start Guide

This Quick Start Guide should enable the user to implement an application program using an HFI (High-Level Language Fieldbus Interface), which operates all controller boards supported by Phoenix Contact. The supported controller boards are listed in "Supported Controller Boards" on page 1-2.

Section 3, "Example Program in C#" uses an example code in C# to illustrate how a high-level language program can be used to access the controller boards supported by Phoenix Contact via the "HFI" library.

The available example programs (see "Available Example Programs in C#" on page 1-3) can be used as a basis and adapted to meet your specific requirements. For programming in Visual Basic (VB), the C# example programs can still be used as a basis, by adapting them to VB. Should you have any questions, please contact Phoenix Contact.

Section 4, "Additional Software" shows how to use an existing bus configuration and additional software to integrate the I/O system connected to the supported hardware in your control program.

1.2 HFI Interface for Data Access in the Field

HFI = **H**igh-Level Language **F**ieldbus **I**nterface

The object-oriented and .NET-capable HFI user interface can be used by a Windows XP-based PC control program to read and control data from the field. I/O signals and diagnostic data can be accessed from every .NET application via a class library. At signal level, the HFI library supports PCI cards with direct INTERBUS master and bus couplers with Ethernet connection and the Ethernet gateway from the Factory Line product range (see also "Supported controller boards" on page 1-2).

The PC control program functions can be integrated easily. All data access is performed via registered variables and diagnostic messages are processed automatically by the classes. In addition, information can be transferred directly from the INTERBUS bus configurator CMD (IBS CMD SWT G4 E, Order No. 2721442).

1.3 System Requirements

Table 1-1 provides an overview of the environment required for HFI 1.1x or HFI 2.0x and the development system that is compatible for each version.

Table 1-1 System requirements for HFI

Product (Setup)	Environment	Development System
HFI 1.1x	Windows XP +SP1, .NET Framework 1.1 + SP1	Microsoft Visual Studio 2003, C# 2003, VB 2003, SharpDevelop (free of charge)
HFI 2.0x	Windows XP +SP1, .NET Framework 2.0	Microsoft Visual Studio 2005, C# 2005, VB 2005, Visual Studio Express (free of charge), SharpDevelop (free of charge)

It is assumed the user has experience in Microsoft Windows operating systems and the listed Microsoft programs.

Example projects are available on the Internet, e.g., at www.codeproject.com and www.csharp.com.

1.4 Supported Controller Boards

Table 1-3 lists all the controller boards supported by the HFI user interface.

Table 1-2 Supported controller boards

Description	Type	Order No.
Controller board for PC systems with PCI bus	IBS PCI SC/I-T	2725260
Controller board for PC systems with PCI 104 bus	IBS PCI 104 SC-T	2737494
Ethernet/Inline bus coupler	FL IL 24 BK-B-PAC	2862327
Ethernet/Inline bus coupler	FL IL 24 BK-PAC	2862314
Inline bus coupler for Ethernet with eight digital inputs and four digital outputs	IL ETH BK DI8 DO4 2TX-PAC	2703981
Inline Block IO module for Ethernet with 16 digital inputs and 16 digital inputs or outputs	ILB ETH 24 DI16 DIO16-2TX	2832962

1.5 Software Requirements

IBS PCI SC I-T, IBS PCI 104 SC-T

In order to work with the HFI interface for these controller boards, the following driver must be installed on your PC: "Win2000_XP_PCI_205.exe" or later.

The driver is available on the "CD PCI DRIVER" CD (Order No. 2985589) or at www.download.phoenixcontact.com in the area for the supported controller board.

Other controller boards

For all other controller boards, the required drivers are installed automatically during installation of the HFI (see Section 2, "Setup for the HFI").

1.6 Available Example Programs in C#

Table 1-3 shows which example program can be used for which controller board.

Table 1-3 Available example programs in C#

Example Name	Function	Supported Controller Boards
HFI Demo CS.sln	Startup of a controller board with INTERBUS startup, I/O data exchange, and PCP communication	IBS PCI SC I-T IBS PCI 104 SC-T FL IBS SC/I-T FL IL 24 BK-B-PAC FL IL 24 BK-PAC IL ETH BK DI8 DO4-2TX-PAC
HFI Demo ILB CS.sln	Startup of an Inline Block IO module with I/O data exchange	ILB ETH 24 DI16 DIO16

1.7 Additional Documentation

The Reference Manual for the HFI user interface is only available as online help. This online help essentially provides an overview of all available classes.

For additional information, please refer to the "Firmware Services and Error Messages" user manual IBS SYS FW G4 UM E (Order No. 2745185).

2 Setup for the HFI

The setup is available on the "CD PCI DRIVER" CD (Order No. 2985589) or at www.download.phoenixcontact.com in the area for the supported controller board.

The installation program generates all the directories required for operation and copies the files for the HFI.

Make sure the required driver is installed on your PC (see "Software Requirements" on page 1-3).

Notes on Software:

– In the Start menu, select "Start... Programs... Phoenix Contact... DotNet Framework..." to access example projects and HFI tools.
– The required assemblies for the libraries are located in the following directory: (..\DotNet Framework...\HFI DotNet\Libraries)
– The file names for the assemblies have the extension (_FX11, _FX20, etc.). This extension indicates the .Net framework for which the relevant assembly is approved. FX is the abbreviation for framework, the subsequent information specifies the framework version (e.g., 11 for 1.1).

Structure of an HFI Application With Possible Controller Boards

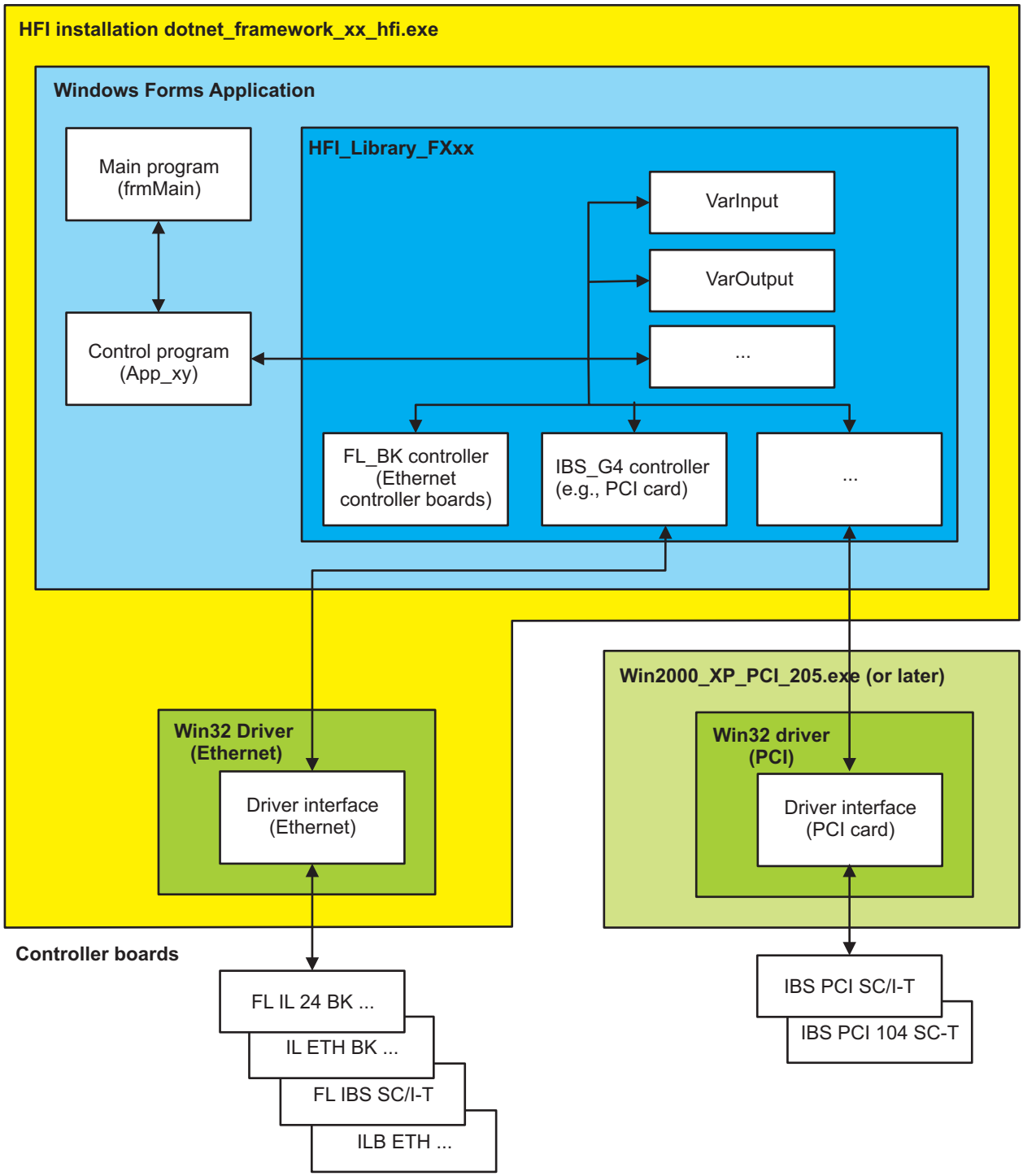


Figure 2-1 Structure of an HFI application

3 Example Program in C#

The example was created using Microsoft Visual Studio. If you are using another development environment, adapt the example accordingly.

For the example, the following configuration was selected:

The FL IL 24 BK-PAC bus coupler is connected to a PC via Ethernet.
The following I/O terminals are connected to the bus coupler:

– IB IL 24 DO 16
– IB IL 24 DO 32
– IB IL 24 DI 16
– IB IL 24 DI 32
– IB IL 24 RS 232

Explanations for the example program are given below, as well as a description of where adaptations can or should be made.

- Open the example project via "Start... Programs... Phoenix Contact... DotNet Framework... HFI Demo CS".

In Solution Explorer, the "References" folder contains the integrated program libraries HFI_Library_FX11 and HFI_Visu_FX11.

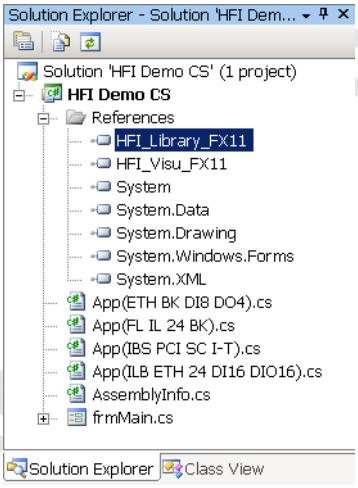


Figure 3-1 Integrated program libraries

- In Solution Explorer, open the source code for class "frmMain.cs".

- Select the "Controller" HFI class, which corresponds to the controller board used. Remove the comment characters (//) for the corresponding entry. The entries for the other controller boards should be commented out. In the example, the FL IL 24 BK-PAC bus coupler is used as the controller board.

```
namespace HFI_Demo
{
    /// <summary>
    /// Summary for frmMain
    /// </summary>
    public class frmMain : System.Windows.Forms.Form
    {

        // Create the instance from a select controller class
        // TODO Please select you controller type
//      App_IBS_PCI_SC_IT          myApplication = new App_IBS_PCI_SC_IT();
//      App_ETH_BK_DI8_DO4         myApplication = new App_ETH_BK_DI8_DO4();
        App_FL_IL_24_BK            myApplication = new App_FL_IL_24_BK();
//      App_ILB_ETH_24_DI16_DIO16  myApplication = new App_ILB_ETH_24_DI16_DIO16();
```

Figure 3-2 Integrated program libraries

- In Solution Explorer, open the class with the example program for your controller board. For the FL IL 24 BK-PAC bus coupler, this is "App(FL IL 24 BK).cs".

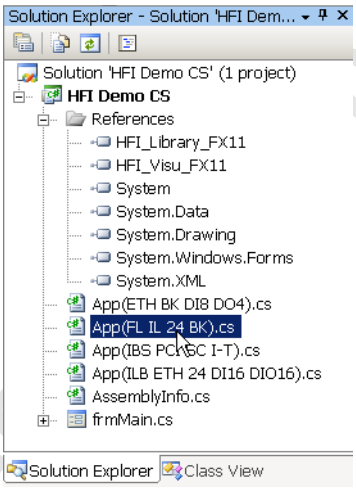


Figure 3-3 Opening the example program for the FL IL 24 BK-PAC

```
namespace HFI_Demo
{
    public sealed class App_FL_IL_24_BK :IDisposable
    {
        *** Variable Declaration ***************************************************

        *** Constructor Declaration ************************************************

        *** Events From the Controller *********************************************

        *** Enable / Disable the Application ***************************************

        *** Get the PCP-Data From the Application **********************************

        *** IDisposable Member *****************************************************

    }
}
```

Figure 3-4 Program parts of the example program

The individual program parts are described below.

3.1 Variable Settings (Variable Declaration)

- In the program, adapt the variable declaration to the bus configuration.

The variable declarations have the following parameters:

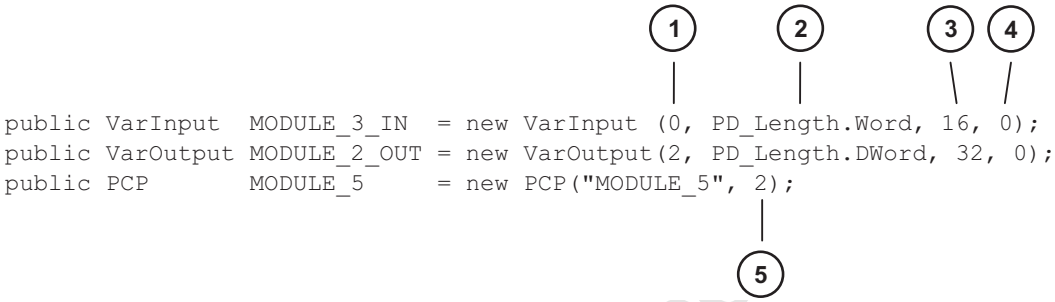


Figure 3-5 Parameters for input and output variables

Key:

1	Byte address	INTERBUS parameter
2	Process data length	INTERBUS parameter
3	Bit length of the process data item	User-defined parameter
4	Bit offset of the process data item	User-defined parameter
5	Communication reference (CR)	INTERBUS parameter PCP

If you know the data for the INTERBUS parameters, enter it at the relevant points. If you do not know the parameters, they can be generated. Use the HFI Device Explorer (see "HFI Device Explorer" on page 4-2) or CMD (see "CMD" on page 4-6).

The user-defined parameters are generally specified by the user. These parameters can be used to address the modules as an overall object or to define individual objects, which comprise one or more bits.

The HFI Code Generator (see "HFI Code Generator" on page 4-8) can be used to generate the source code. However, single-bit addressing is not supported here. The modules are addressed as an overall object with the generated start address.

- Adapt the variable declaration for the input variables.

```
// Create the variables for the input data
// First input terminal DI 16
public VarInput            IN_Bit_0       = new VarInput(0, PD_Length.Word, 1, 0);
public VarInput            IN_Bit_1       = new VarInput(0, PD_Length.Word, 1, 1);
public VarInput            IN_Variable    = new VarInput(0, PD_Length.Word, 12, 4);

// Second input terminal DI 32
public VarInput            IN_ByteArray   = new VarInput(2, 4);

// PCP terminal inputs (RS232 terminal)
public VarInput            IN_RS232_1     = new VarInput(8, PD_Length.Word, 16, 0);
```

Figure 3-6 Input variables

- Adapt the variable declaration for the output variables.

```
// Create the variables for the output data
// First output terminal DO 16
public VarOutput           OUT_Bit_0     = new VarOutput(0, PD_Length.Word, 1, 0);
public VarOutput           OUT_Bit_1     = new VarOutput(0, PD_Length.Word, 1, 1);
public VarOutput           OUT_Variable  = new VarOutput(0, PD_Length.Word, 12, 4);

// Second output terminal DO 32
public VarOutput           OUT_ByteArray = new VarOutput(2, 4);


// PCP terminal outputs (RS232 terminal)
public VarOutput          OUT_RS232_1    = new VarOutput(8, PD_Length.Word, 16, 0);
```

Figure 3-7 Output variables

- Adapt the variable declaration for the variables for PCP communication.

```
// Create the variables for the PCP communication CR (RS232 terminal)
public PCP                 PCP_RS232_1    = new PCP("RS232_1", 2);

private byte[] _pcpReadBuffer = new byte[0];
private byte[] _pcpWriteBuffer = new byte[0];
```

Figure 3-8 Variables for PCP communication

3.2 Settings for the "Controller" Class (Constructor Declaration)

The settings for the "Controller" class are made in the constructor.

- Adapt the settings.

```
#region *** Constructor declaration ***********************************************

/// <summary>
/// Constructor
/// </summary>
public App_FL_IL_24_BK()
{
   // Create the controller with a name
   Controller = new Controller_FL_BK("FL IL 24 BK");

   // Settings for the controller
   Controller.Description = "FL IL 24 BK for Demonstaration";

   Controller.Startup     = ControllerStartup.PhysicalConfiguration;

   Controller.UpdateProcessDataCycleTime = 20;
   Controller.UpdateMailboxTime          = 50;

   // The Controller.Configuration property contains special configurations for the controller
//   Controller.Configuration.ControlCPU_Load        = false;
//   Controller.Configuration.DNS_NameResolution     = true;
//   Controller.Configuration.ErrLogActivate         = true;
//   Controller.Configuration.ErrLogFilename         = @"c:\Test.log";
   Controller.Configuration.ExpertModeActivate     = false;
//   Controller.Configuration.GetVersionInfo         = false;
//   Controller.Configuration.UpdateControllerState  = 100;
```

Figure 3-9 Settings for the "Controller" class

- Set the start behavior (see also Table 4-1 "Bus configuration options" on page 4-1). In the example, "PhysicalConfiguration" is selected as the start behavior.
- Set the process data cycle time (ProcessDataCycleTime; 20 ms in the example).
- Set the update time for the mailbox (UpdateMailboxTime; 50 ms in the example).
- Set the operating mode (see also "Note on "ExpertModeActivate"" on page 3-6).

If no changes are made, the default values are set. If you remove the comment characters (//), this activates or changes the settings.

Note on "ExpertModeActivate"

To work with the HFI, "Expert Mode" must be activated for all controller boards.

If "Expert Mode" is not activated, errors will occur during startup.

Please note the following:

1. FL IL 24 BK-PAC and FL IL 24 BK-B-PAC bus couplers
   In the program code, deactivate "Expert Mode" ("false"). Activate it instead via the HFI Device Explorer (see "HFI Device Explorer" on page 4-2).
2. IL ETH BK DI8 DO4-2TX-PAC bus coupler
   Activate "Expert Mode" either in the program code ("true" = default setting) or via the HFI Device Explorer.
3. All other controller boards
   Activate "Expert Mode" in the program code ("true" = default setting).

In the example, a FL IL 24 BK-PAC is used, which is why "Expert Mode" is deactivated in the illustrated example program code.

Adding variables

In the following program part, the variables, which were created and addressed above, are added to the "Controller" class and therefore registered.

```
// Add input variables to the controller
Controller.AddObject(IN_Bit_0);
Controller.AddObject(IN_Bit_1);
Controller.AddObject(IN_Variable);
Controller.AddObject(IN_ByteArray);

Controller.AddObject(IN_RS232_1);


// Add output variables to the controller
Controller.AddObject(OUT_Bit_0);
Controller.AddObject(OUT_Bit_1);
Controller.AddObject(OUT_Variable);
Controller.AddObject(OUT_ByteArray);

Controller.AddObject(OUT_RS232_1);

// Add PCP objects to the controller
Controller.AddObject(PCP_RS232_1.ControllerConnection);
```

Figure 3-10 Adding variables

Creating callbacks

In the following program part, callbacks (event-controlled functions) are created.

```
// Callbacks for the controller

// Called once for each bus cycle
Controller.OnUpdateProcessData  +=new UpdateProcessDataHandler(Controller_OnUpdateProcessData);

// Called once for each mailbox cycle
Controller.OnUpdateMailbox      +=new UpdateMailboxHandler(Controller_OnUpdateMailbox);

// Called whenever an error occurs in the controller object
Controller.OnDiagnostic         +=new DiagnosticHandler(Controller_OnDiagnostic);


// Events from PCP_2
PCP_RS232_1.OnEnableReady                += new EnableReadyHandler(PCP_RS232_1_OnEnableReady);
PCP_RS232_1.OnReadConfirmationReceived   += new ConfirmationReceiveHandler(PCP_RS232_1_ReadConfirmationReceived);
PCP_RS232_1.OnWriteConfirmationReceived  += new ConfirmationReceiveHandler(PCP_RS232_1_WriteConfirmationReceived);
PCP_RS232_1.OnDiagnostic                 += new DiagnosticHandler(PCP_RS232_1_OnDiagnostic);
```

Figure 3-11 Creating callbacks

3.3 Events From the Controller

Notes on Events

– Only register events, which are required.
– Do not create blocking programming ("while") or integrate waiting times ("sleep").
– Always use parallel threads or timers to access "Forms", databases, etc.

Blocking an event blocks the complete "Controller" class and therefore the complete application.

OnUpdateProcessData

The "OnUpdateProcessData" event is called cyclically at the interval set for the process data cycle time (20 ms).
In the "ProcessDataEvent" function registered in the "OnUpdateProcessData" event (see Figure 3-11 on page 3-7), the process data is processed.

```
#region *** Events From the Controller *************************

/// <summary>
/// Called once for each bus cycle
/// </summary>
/// <param name="state"></param>
private void Controller_OnUpdateProcessData(object Sender)
{
   // TODO insert your process data handling (application) here

   // Test application for a counter
   if (OUT_Variable.Value < OUT_Variable.MaxValue)
   {
      OUT_Variable.Value++;
   }
   else
   {
      OUT_Variable.Value = OUT_Variable.MinValue;
   }

}
```

Figure 3-12 ProcessDataEvent

OnUpdateMailbox

The "OnUpdateMailbox" event is called cyclically at the interval set for the mailbox update time (50 ms).
In the "MailboxDataEvent" function registered in the "OnUpdateMailbox" event (see Figure 3-11 on page 3-7), the PCP device is activated or deactivated.

```
/// <summary>
/// Called once for each mailbox cycle
/// </summary>
/// <param name="Sender"></param>
private void Controller_OnUpdateMailbox(object Sender)
{
   // Enable/Disable the PCP Device
   if (Controller.IBS_Diag.StatusRegister.RUN)
   {
      if (!PCP_RS232_1.Ready && !PCP_RS232_1.Error)
         PCP_RS232_1.Enable();
   }
   else
   {
      if (PCP_RS232_1.Ready || PCP_RS232_1.Error)
         PCP_RS232_1.Disable();
   }

   // TODO insert your mailbox handling here (is called once for each MX cycle)
}
```

Figure 3-13 MailboxDataEvent

OnDiagnostic (Controller)

The "OnDiagnostic" event is called on a change in the diagnostic status of the "Controller" class.
The "DiagnosticRun" function registered in the "OnDiagnostic" event (see Figure 3-11 on page 3-7) displays the current diagnostic message in a non-blocking message box.

```
/// <summary>
///  Called whenever an error occurs in the controller object
/// </summary>
/// <param name="Sender"></param>
/// <param name="Diagnostic"></param>
private void Controller_OnDiagnostic(object Sender, DiagnosticArgs DiagnosticCode)
{
   // Shows each error message
   Util.MessageBoxShow(Sender, Diagnostic);

   // TODO your error handling can be inserted here
}
```

Figure 3-14 DiagnosticRun

OnReadConfirmation Received

The "OnReadConfirmationReceived" event is called if there is PCP data available for processing.
The "PCP_RS232_1_ReadConfirmationReceived" function registered in the "OnReadConfirmationReceived" event (see Figure 3-11 on page 3-7) is used to transfer the PCP data to a data memory for further processing.

```
/// <summary>
/// Called for each successfull read confirmation
/// </summary>
/// <param name="Sender"></param>
/// <param name="Data"></param>
private void PCP_RS232_1_ReadConfirmationReceived(object Sender, byte[] Data)
{
   // TODO insert your code here
   lock(_pcpReadBuffer)
   {
      _pcpReadBuffer = new Byte[Data.Length];
      _pcpReadBuffer = Data;
   }
}
```

Figure 3-15 PCP_RS232_1_ReadConfirmationReceived

OnWriteConfirmation Received

The "OnWriteConfirmationReceived" event is called when the PCP device confirms a write service.
The "Data" data for the "PCP_RS232_1_WriteConfirmationReceived" function registered in the "OnWriteConfirmationReceived" event (see Figure 3-11 on page 3-7) can be used to determine whether the write service was successful or not.

```
/// <summary>
/// Called for each successfull write confirmation
/// </summary>
/// <param name="Sender"></param>
/// <param name="Data"></param>
private void PCP_RS232_1_WriteConfirmationReceived(object Sender, byte[] Data)
{
    // TODO insert your code here
   lock(_pcpWriteBuffer)
   {
      _pcpWriteBuffer = new Byte[Data.Length];
      _pcpWriteBuffer = Data;
   }
}
```

Figure 3-16 PCP_RS232_1_WriteConfirmationReceived

OnDiagnostic (PCP)

The "OnDiagnostic" event is called on a change in the PCP status of a PCP object.
The "PCP_RS232_1_OnDiagnostic" function registered in the "OnDiagnostic" event (see Figure 3-11 on page 3-7) displays the current diagnostic message in a non-blocking message box.

```
/// <summary>
/// Called whenever an error occurs in the pcp object
/// </summary>
/// <param name="Sender"></param>
/// <param name="Diagnostic"></param>
private void PCP_RS232_1_OnDiagnostic(object Sender, DiagnosticArgs Diagnostic)
{
   // Shows each diagnostic message
   Util.MessageBoxShow(Sender, Diagnostic);

   // Your diagnostic handling can be inserted here
}
```

Figure 3-17 PCP_RS232_1_OnDiagnostic

OnEnableReady

The "OnEnableReady" event is called when a connection ("Initiate") has been established with the PCP device.

3.4 Activating/Deactivating the Control Program (Enable/Disable the Application)

Enable

In the following program part, a method is set for **activating** the control program.

```
#region *** Enable / Disable the Application ****************

/// <summary>
/// This method enables the controller and the PCP devices
/// </summary>
public void Enable()
{
   Controller.Enable();
}
```

Figure 3-18 Activating the control program

Disable

In the following program part, a method is set for **deactivating** the control program.

```
/// <summary>
/// This method disables the PCP devices and the controller
/// </summary>
public void Disable()
{
   // Disable the PCP devices
   PCP_RS232_1.Disable();

   // Waiting for the disconnection from the PCP terminal
   System.Threading.Thread.Sleep(Controller.UpdateMailboxTime * 4);

   // Disables the controller
   Controller.Disable();
}

#endregion
```

Figure 3-19 Deactivating the control program

When deactivating the control program, proceed as follows:

- First deactivate the PCP devices by calling the "Disable" method.
- Wait until the connections are aborted by the device.
  Observe a duration of approximately four times the set mailbox update time (UpdateMailboxTime * 4).
- Deactivate the "Controller" class.

3.5 Function for PCP Data Exchange (Get the PCP Data From the Application)

The following program part implements PCP data exchange between the control program and the program user interface.

Do not write directly from the "OnReadConfirmationReceived" event to the program user interface (Form). Implement data exchange using a parallel thread or a parallel timer.

A timer is used in the example.

```
#region *** Get the PCP-Data from the application *****

/// <summary>
/// Get the PCP Read Buffer
/// </summary>
public Byte[] PCP_ReadData
{
   get
   {
      lock(_pcpReadBuffer)
      {
         return _pcpReadBuffer;
      }
   }
}
```

Figure 3-20 PCP data exchange with the program user interface

3.6 Closing the Application Program (IDisposable Member)

The following program part exits the control program. This ensures that all connections are aborted and all processes are exited.

```
#region *** IDisposable Member *****

public void Dispose()
{
   Disable();
   Controller.Dispose();
}

#endregion
```

Figure 3-21 Exiting the program

3.7 Function for Data Exchange (Update the Data on the Form)

- Switch to the "frm.Main.cs" class.

The following program part implements data exchange between the control program and the program user interface.

Do not write directly from the events to the program user interface (Form). Implement data exchange using a parallel thread or a parallel timer.

A timer is used in the example.

```
#region *** Update the Data on the Form ***********************************

/// <summary>
/// Update the form
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void tmrMainFormUpdate_Tick(object sender, System.EventArgs e)
{
   // Show the controller state
   cbxReady.Checked     = myApplication.Controller.Ready;
   cbxError.Checked     = myApplication.Controller.Error;

   ...


}
```

Figure 3-22 Data exchange with the program user interface

3.8 Executing the Example Program

The main program points have now been considered and/or adapted. You can now execute and test the program. Translate the program and start it. The program user interface is opened.

- In the "Controller Handling" area, enter the IP address of the controller board.
- Start the "Controller" class by clicking on "Enable".

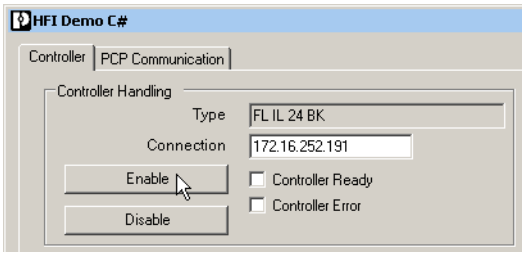


Figure 3-23 Setting the IP address and activating the "Controller" class

Figure 3-24 shows the entire user interface for the example program.

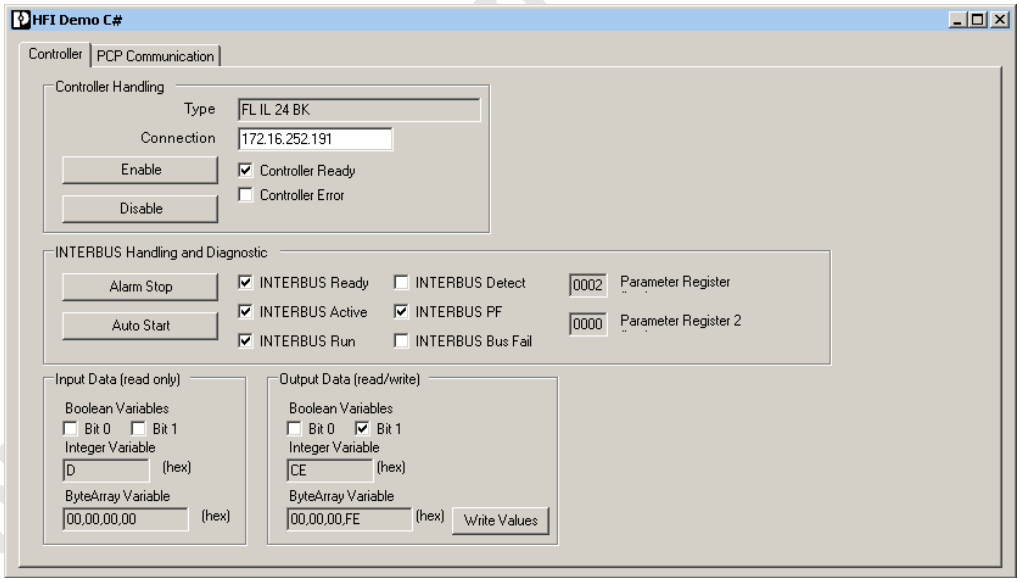


Figure 3-24 User interface for the example program

In the "Controller Handling" area, the "Controller Ready" checkbox indicates that the "Controller" class has been started successfully.

The "INTERBUS Handling and Diagnostic" area shows the behavior of the bus, e.g., for the "Alarm Stop" or "Auto Start" actions.

The "Input Data" and "Output Data" areas can be used to read the status of inputs or write outputs.

- To write output data, activate the fields for the bit variables or enter "Integer" or "ByteArray" variables.
- Then click on "Write Values".

The second page of the user interface is where PCP communication is mapped.

PCP communication is activated/deactivated automatically by the control program (see "MailboxDataEvent" on page 3-9).

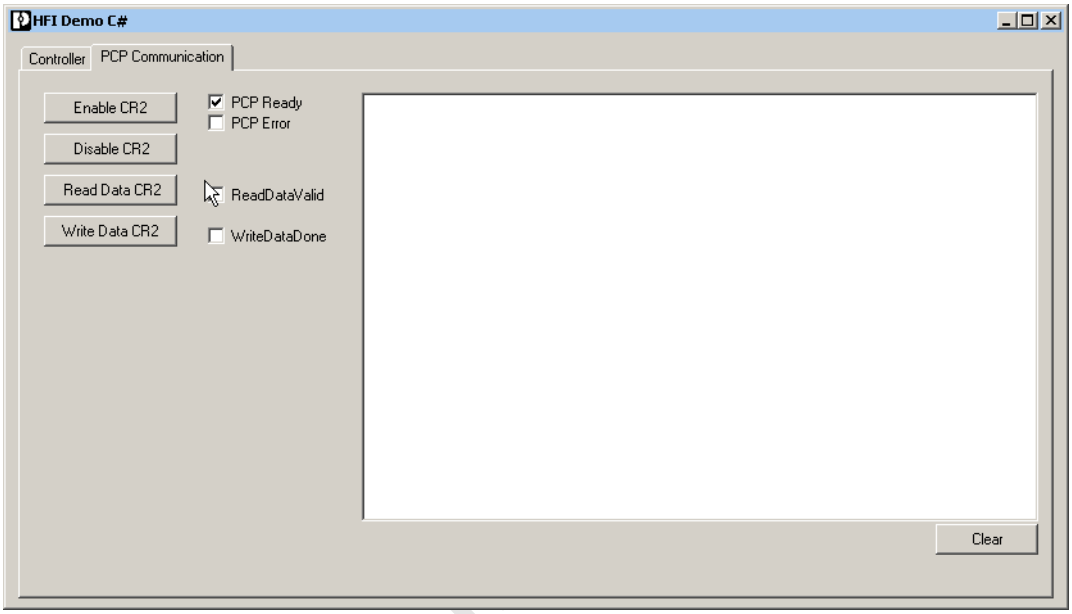


Figure 3-25 User interface for the example program: "PCP Communication" tab

- Click on "Read Data CR2".
  Data from the IB IL RS 232 terminal is read.

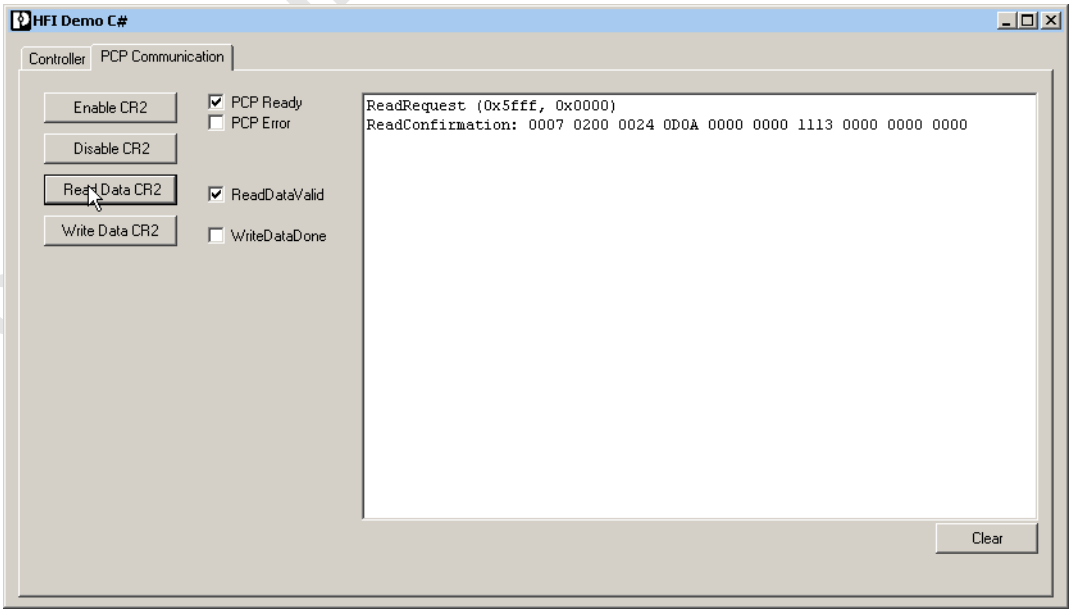


Figure 3-26 PCP data read

- Click on "Write Data CR2".
  The "Baud-Rate" parameter for the IB IL RS 232 terminal is initialized at 19200.

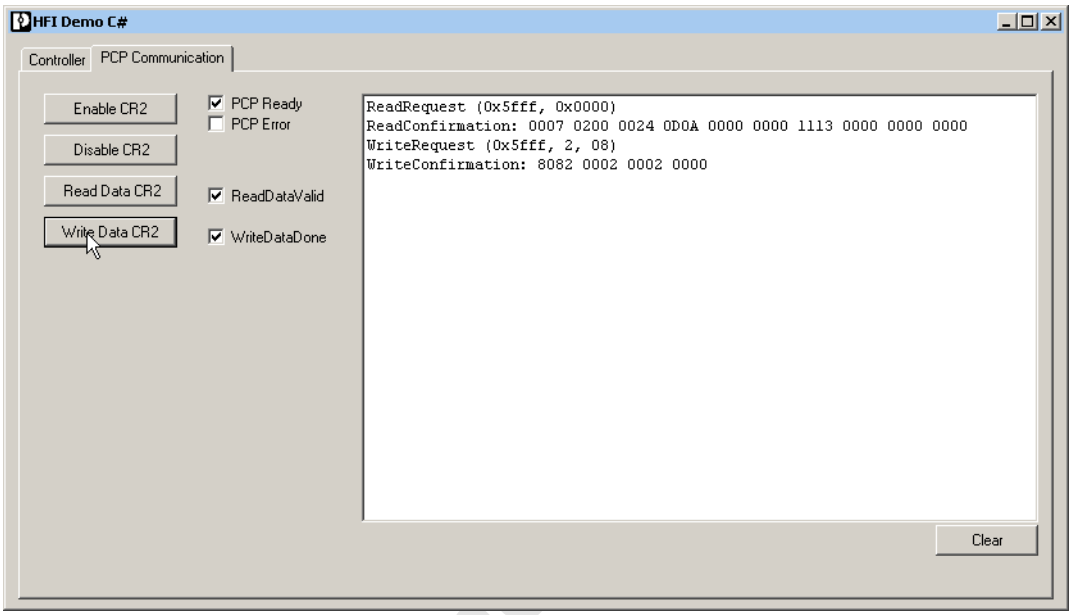


Figure 3-27 PCP data written

- Click on "Read Data CR2". Reading the data again shows the change made by writing.
  The first word contains the new setting (0008).

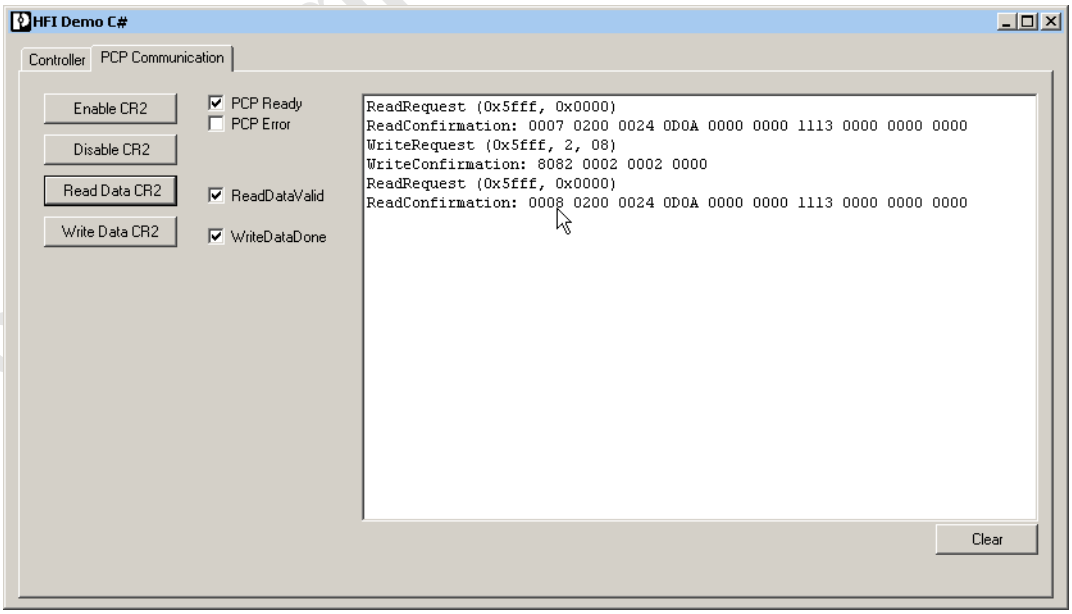


Figure 3-28 PCP data read again

4 Additional Software

4.1 Bus Configuration

Depending on the controller board used, there are various options for configuring the bus.

Table 4-1 Bus configuration options

Controller Board	Logical		SVC File	Physical
	CMD	Plug and Play		
IBS PCI SC/I-T	Yes	No	Yes	Yes
IBS PCI 104 SC-T	Yes	No	Yes	Yes
FL IBS SC/I-T	Yes	No	Yes	Yes
FL IL 24 BK-B-PAC	No	Yes	No	Yes
FL IL 24 BK-PAC	No	Yes	No	Yes
IL ETH BK DI8 DO4 2TX-PAC	No	Yes	No	Yes
ILB ETH 24 DI16 DIO16-2TX	No	No	No	Yes

Logical configuration (LogicalConfiguration)

- Via CMD

For a logical bus configuration via CMD, the controller board must have been parameterized at least once with CMD and the parameterization must have been saved.

- Via plug and play

In plug and play mode, the controller board reads the connected bus configuration and stores this configuration permanently in the memory. This stored configuration is used during startup with a logical configuration.

Configuration via SVC file (SvcFileConfiguration)

For parameterization with the generated SVC file (service file), the control PC writes the firmware services contained in the SVC file to the controller board.
This option is ideal if CMD is not available on the control PC (e.g., for a series-production machine).

Physical configuration (PhysicalConfiguration)

For a physical bus configuration, CMD and plug and play mode are not required. The controller activates the connected bus configuration as the valid configuration frame.
This option is primarily used for tests during the configuration phase. This means that the control program can be restarted again immediately following a change in the bus configuration, without having to modify the CMD configuration every time.
The user must ensure that the process data addressing corresponds to the existing bus configuration.

4.2 Process Data Addressing

In order to generate a CSV file with process data addressing, the following software tools can be used depending on the controller board used:

– HFI Device Explorer, which is installed with the HFI setup
– CMD, which must be installed separately (IBS CMD SWT G4 E, Order No. 2721442)

Table 4-2 Software tool for process data addressing depending on the controller board

Type	CMD	HFI Device Explorer
IBS PCI SC/I-T	Yes	No
IBS PCI 104 SC-T	Yes	No
FL IBS SC/I-T	Yes	No
FL IL 24 BK-B-PAC	No	Yes
FL IL 24 BK-PAC	No	Yes
IL ETH BK DI8 DO4 2TX-PAC	No	Yes
ILB ETH 24 DI16 DIO16-2TX	No	No

For Inline Block IO module addressing, please refer to the corresponding data sheet.

For information on further processing of data in the HFI Code Generator, please refer to "HFI Code Generator" on page 4-8.

4.3 HFI Device Explorer

The HFI Device Explorer tool can read the connected bus configuration of a supported controller board.

Table 4-3 Controller boards supported by the HFI Device Explorer

Type
FL IL 24 BK-B-PAC
FL IL 24 BK-PAC
IL ETH BK DI8 DO4-2TX-PAC

Configuration data can then be entered directly in your development environment or written to a CSV file.

- Open the HFI Device Explorer.
- Click on "Add Device" or "Edit Device" to open the "Edit Device Parameter" window.
- Enter the device name and IP address.
- Set the operating mode for the controller board.

From the three possible controller board operating modes, the HFI requires "Expert Mode". For the controller boards listed in Table 4-3, there are various options for activating this operating mode:

1. FL IL 24 BK-PAC and FL IL 24 BK-B-PAC bus couplers
   Activate "Expert Mode" via the HFI Device Explorer.
   In the program code of the HFI, deactivate "Expert Mode" ("false").
2. IL ETH BK DI8 DO4-2TX-PAC bus coupler

   – Activate "Expert Mode" via the HFI Device Explorer.

   Or

   – Activate "Default Mode" via the HFI Device Explorer and activate "Expert Mode" ("true" = default setting) in the program code of the HFI.

The operating mode set via the HFI Device Explorer is stored permanently on the bus coupler.

For the settings in the program code of the HFI, see "Settings for the "Controller" Class (Constructor Declaration)" on page 3-6.

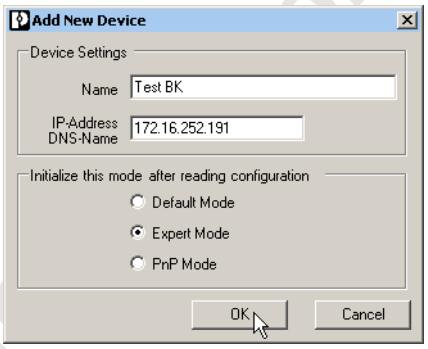


Figure 4-1 HFI Device Explorer: "Add New Device / Edit Device Parameter" window

In "PnP Mode" the HFI cannot be used to access the bus coupler. Do **not** select this operating mode.

- Read the bus configuration by clicking on "Read Bus Configuration".

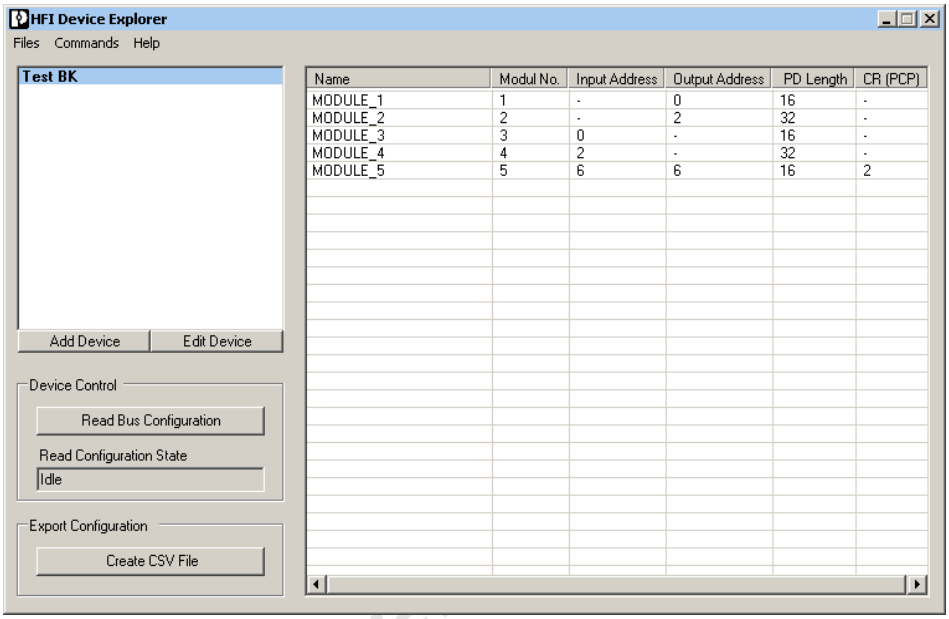


Figure 4-2 HFI Device Explorer

The information displayed for the variables can be entered in the variable declaration for the program. Figure 4-3 shows the relationship between the data in the HFI Device Explorer (A) and in the example program (B).

A

① ②③ ④⑤ ⑥ ⑦⑧

Name	Modul No.	Input Address	Output Address	PD Length	CR (PCP)
MODULE_1	1	-	0	16	-
MODULE_2	2	-	2	32	-
MODULE_3	3	0	-	16	-
MODULE_4	4	2	-	32	-
MODULE_5	5	6	6	16	2

B

```
#region *** Create input variables
                        ①                  ②     ③     ⑥
public VarInput    MODULE_3_IN   = new VarInput(0,PD_Length.Word,16,0);
public VarInput    MODULE_4_IN   = new VarInput(2,PD_Length.DWord,32,0);
public VarInput    MODULE_5_IN   = new VarInput(6,PD_Length.Word,16,0);

#endregion

#region *** Create output variables
                        ①                  ④     ⑤     ⑥
public VarOutput   MODULE_1_OUT = new VarOutput(0,PD_Length.Word,16,0);
public VarOutput   MODULE_2_OUT = new VarOutput(2,PD_Length.DWord,32,0);
public VarOutput   MODULE_5_OUT = new VarOutput(6,PD_Length.Word,16,0);

#endregion

#region *** Create PCP variables
                        ①                  ⑦                  ⑧
public PCP         MODULE_5      = new PCP("MODULE_5", 2);

#endregion
```

Figure 4-3 Variables in the HFI Device Explorer and in Visual Studio

The variables can also be used to generate a CSV file. This can then be further processed in the HFI Code Generator.

- Click on "Create CSV File" to create a CSV file.

For information on further processing in the HFI Code Generator, please refer to "HFI Code Generator" on page 4-8.

4.4 CMD

CMD (IBS CMD SWT G4 E, Order No. 2721442) can be used to read the connected bus configuration from the supported controller boards.

Table 4-4 Controller boards supported by CMD

Type
IBS PCI SC/I-T
FL IBS SC/I-T

Configuration data (start address and process data length) can then be entered directly in your development environment or written to a CSV file.

To create a project, refer to the documentation for CMD.

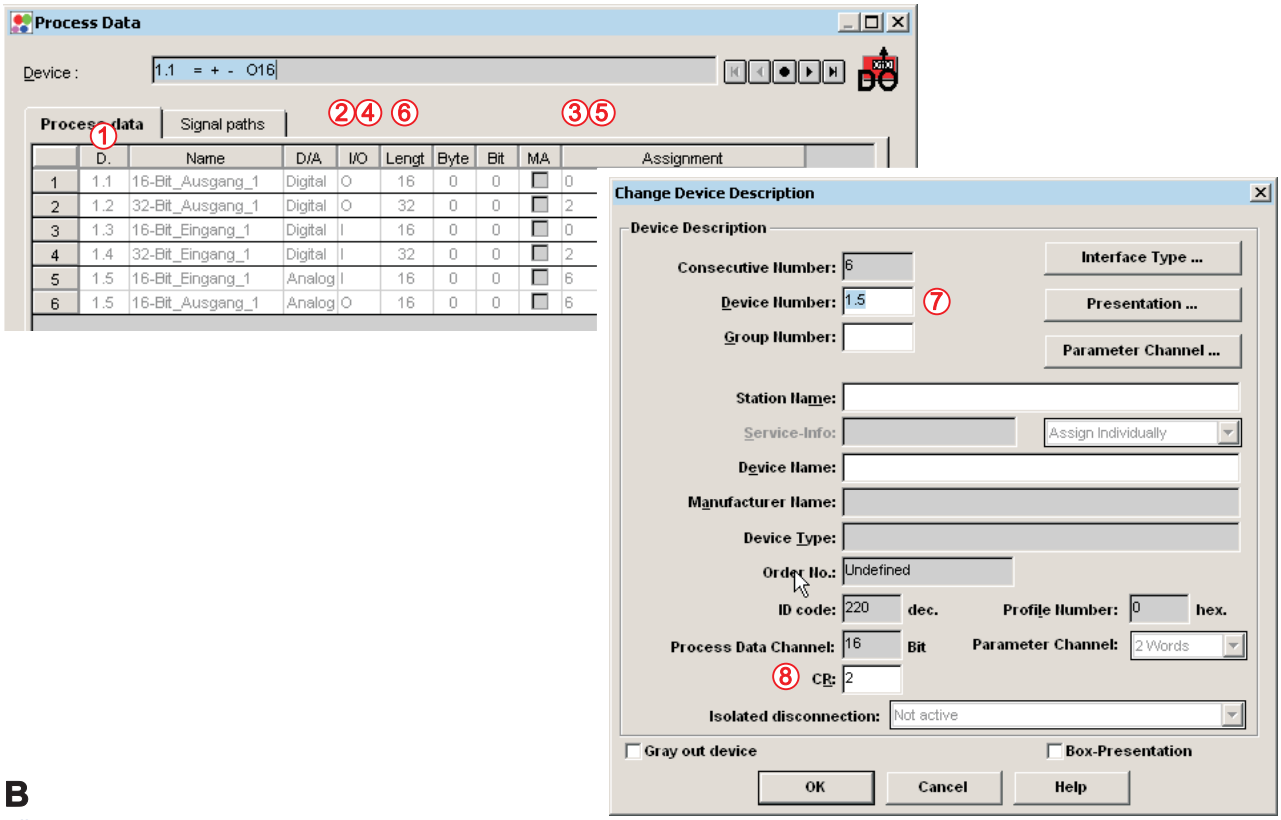
Proceed as follows:

- Start CMD.
- Select the desired controller board.
- Set the communication path.
- Read the bus configuration.
- For the assignment of process data, select "Auto-Address... Startup without... System coupler startup without..." in the Process Data dialog box.
- If PCP devices are present: assign names for PCP devices.
- Set bus startup to "Startup without preprocessing".
  Always select "Activate configuration frame" and "Start data transmission".
- Execute the parameterization as "Startup without preprocessing".
- Save the project under the desired name.
- Save the project to the Flash card of the PCI card.
  To do this, right-click on "Parameterization Memory" to access the context menu and select "Write".
  When asked "Enable read back of the current project file?", select "No".
- Generate a SVC file (for the bus configuration).
  To do this, right-click on "Parameterization Memory" to access the context menu and select "Write ASCII File... INTERBUS Data (*.SVC)...".
- Save the SVC file.
- Generate a CSV file (for the code generator).
  To do this, right-click on "Parameterization Memory" to access the context menu and select "Write ASCII File... Project Data (*.CSV)...".
  Select all options apart from "Comment".
- Save the CSV file.

The generated CSV file is required for code generation.

The information displayed for the variables can be entered in the variable declaration for the program. Figure 4-3 shows the relationship between the data in the HFI Device Explorer (A) and in the example program (B).

A



B

```
#region *** Create input variables
                        ①                   ②  ③      ⑥
public VarInput    MODULE_3_IN   = new VarInput(0,PD_Length.Word,16,0);
public VarInput    MODULE_4_IN   = new VarInput(2,PD_Length.DWord,32,0);
public VarInput    MODULE_5_IN   = new VarInput(6,PD_Length.Word,16,0);

#endregion

#region *** Create output variables
                        ①                   ④  ⑤      ⑥
public VarOutput   MODULE_1_OUT = new VarOutput(0,PD_Length.Word,16,0);
public VarOutput   MODULE_2_OUT = new VarOutput(2,PD_Length.DWord,32,0);
public VarOutput   MODULE_5_OUT = new VarOutput(6,PD_Length.Word,16,0);

#endregion

#region *** Create PCP variables
                        ①                   ⑦          ⑧
public PCP         MODULE_5     = new PCP("MODULE_5", 2);

#endregion
```

Figure 4-4 INTERBUS parameters in CMD

4.5 HFI Code Generator

The HFI Code Generator tool uses a CSV file and a selected template to create an operational application with all the variables included in the CSV file.

The CSV file is generated either by the HFI Device Explorer or by CMD.

- Open the HFI Code Generator and follow the instructions.

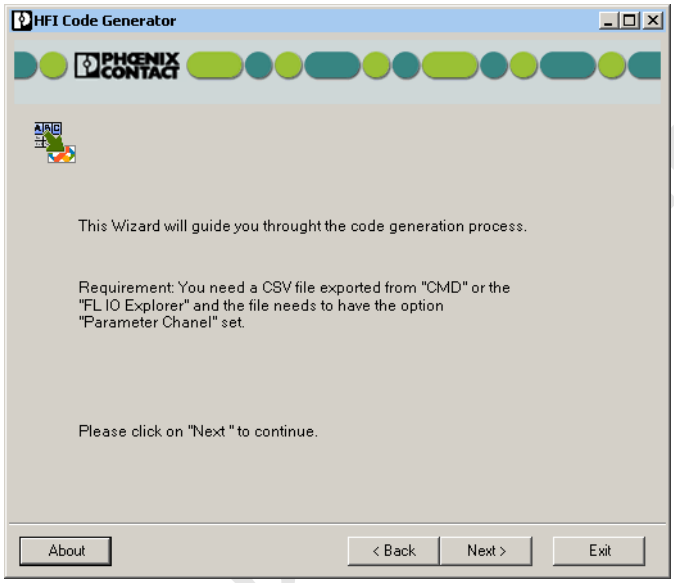


Figure 4-5 HFI Code Generator

In the menu, select the checkboxes for the data that you require.

- Clock on "Read CSV File".

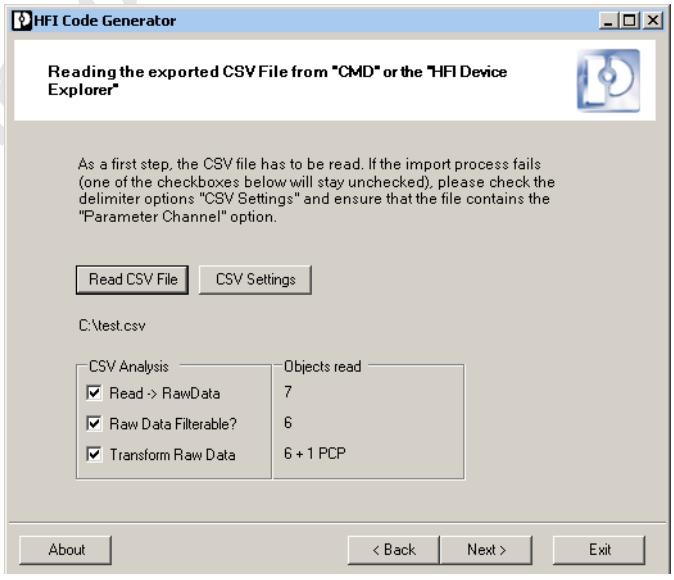


Figure 4-6 Read CSV file

- Select the template (e.g., "VS2003 CS (FL IL 24 BK)").
- Enter the IP address.
- Specify whether you want to generate a complete project (Generate Project) or only the variables (Generate Variables).
  If a project already exists, you only need to generate the variables. In this case a window opens following generation, which displays all generated variables. They can then be copied from this window for further processing in a project.
- Confirm your entries with "Next".

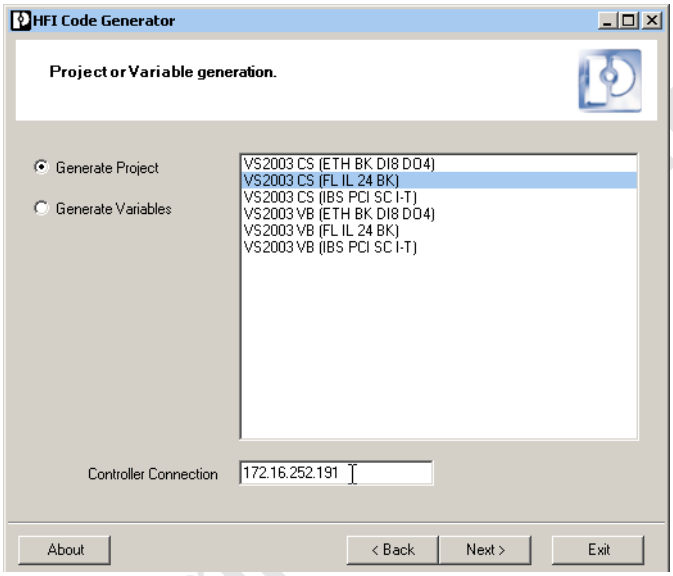


Figure 4-7 Template and IP address for the FL IL 24 BK-PAC

- In the window that opens, select the path for the CSV file.

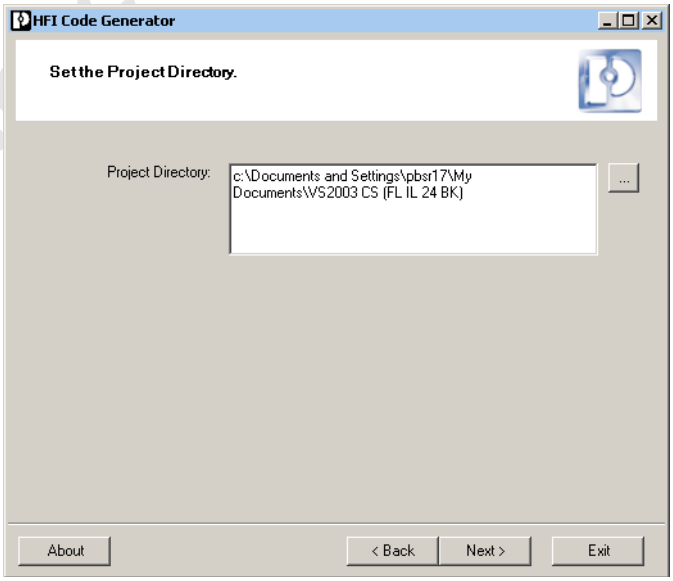


Figure 4-8 Select path

- In the window that opens, click on "Generate" and generate the source code for an example project adapted to your controller board.

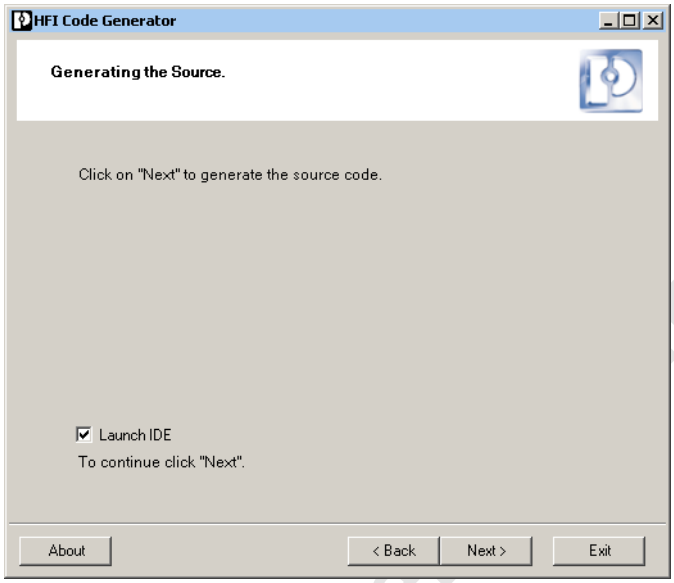


Figure 4-9 Select path

- Open the created application with your development system.

You can add your application program to the generated INTERBUS program part.

To create your application, refer to the documentation for the development system used.

4.6 HFI Controls

4.6.1 Controls for the Application Program

Predefined controls provide quick and easy access to the key functions of the HFI. The controls provide user-friendly diagnostic and test options, e.g., for the Service menu in your application.

Just a few lines of code are required to start up or test a "Controller" class. The available example programs illustrate clearly how the controls and the HFI can be used.

To use the controls in your application program, insert a reference to the "HFI_Visu" component in your project.

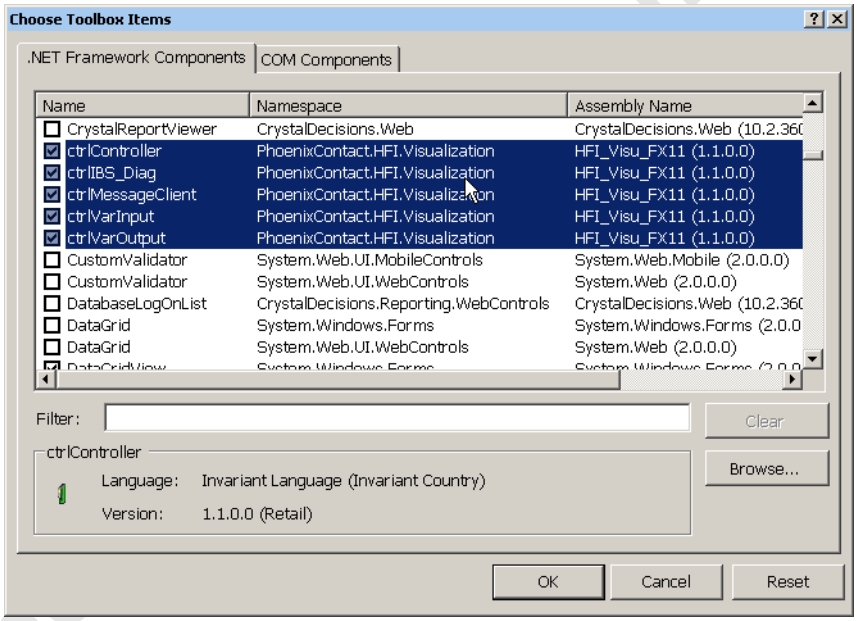


Figure 4-10 HFI controls

4.6.2 Functions of the Controls

ctrlController

Read and operate the controller

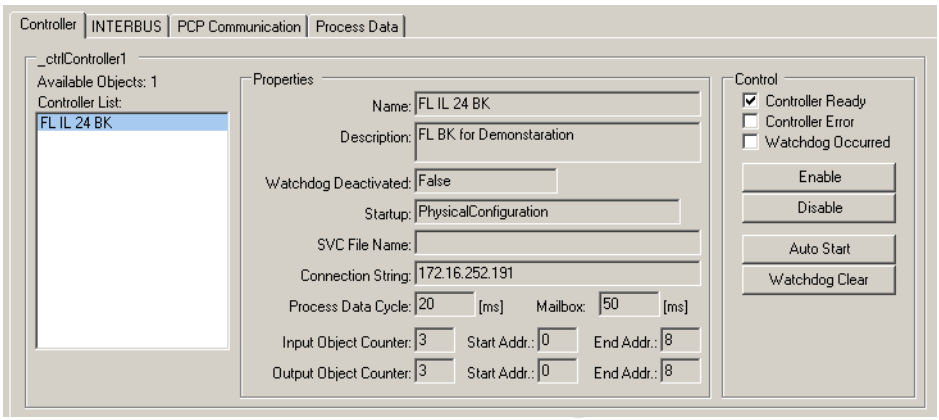


Figure 4-11 Controls: ctrlController

ctrlIBS_Diag

INTERBUS diagnostics and bus handling

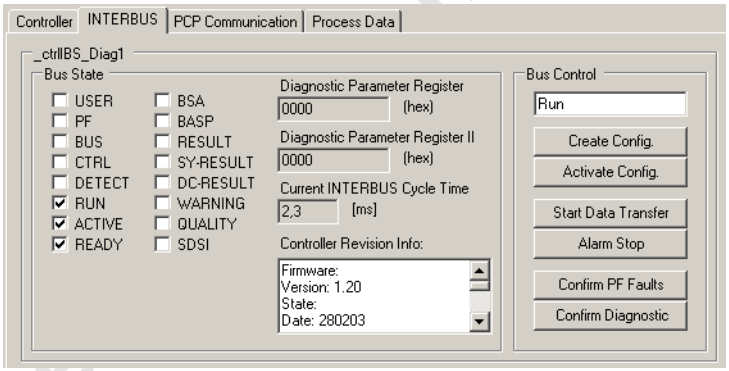


Figure 4-12 Controls: ctrlIBS_Diag

ctrlMessageClient Read PCP and firmware telegrams in the active application

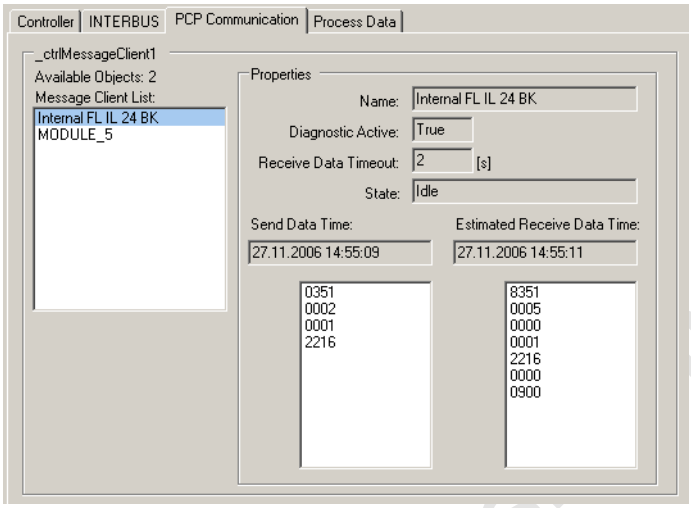


Figure 4-13 Controls: ctrlMessageClient

ctrlVarInput Read the properties of an input object (see Figure 4-14)

ctrlVarOutput Read and write the properties of an output object

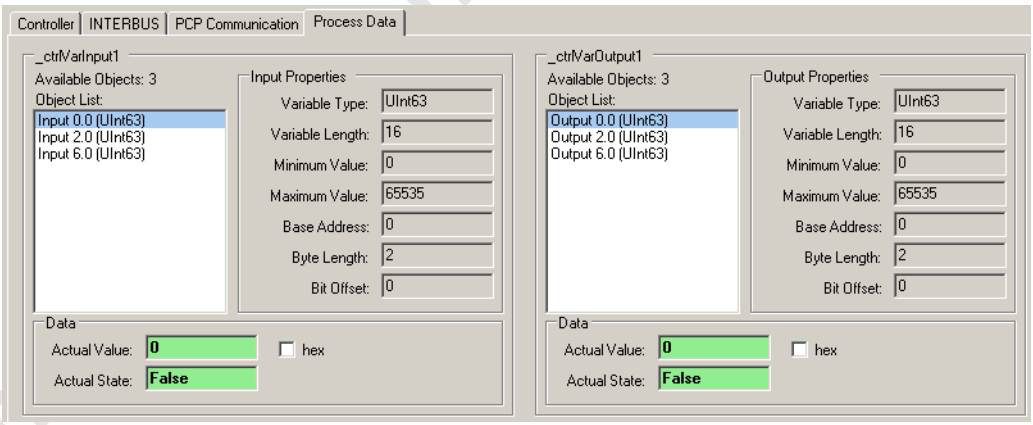


Figure 4-14 Controls: ctrlVarInput, ctrlVarOutput

When the "EditActivate" property is set, the "Actual Value:" output variable value of the selected output object can be edited.

5 Remote Debugging

Initial debugging can often be completed on the local development computer. However, since some problems only occur in the test or production environment, debugging within this environment is also required.

Microsoft provides the Remote Debugger as part of Visual Studio .Net. It can be used to debug an application on another computer.

The information below is provided by the company Microsoft Corporation.

5.1 Remote Debug Monitor

In order to work with the Remote Debug Monitor, install the Machine Debug Manager via the Visual Studio .Net setup. You can either install a full version of Visual Studio .Net or select "Remote Components Setup" in the main menu of the installation routine. Two options are available here:

– Native Remote Debugging:
  Installs components, which enable a debugger to establish a connection exclusively for debugging native code.
– Full Remote Debugging:
  Installs components, which enable a debugger to establish a connection for debugging:
  – Native code
  – Managed code, which is executed in the CLR (Common Language Runtime)
  – Scripts (VB script or JScript)

  If SQL Server is installed on the computer, components for remote SQL debugging are also installed.

If you want to debug C# or VB code, select the second option. This installs all the files required for remote debugging on the system.

As soon as the components for remote debugging are installed, set the system access rights to enable sufficient access.

– Debugging a process from another user:
  You require administrator rights for the computer on which the process is running. This is true whether you are directly accessing a user's application or working with a web application, which accesses the aspnet_wp.exe process.
– Debugging your own process:
  You must be the administrator or a member of the "Debugger Users" group.

If you are working with your own code or process, you can simply add your name to the "Debugger Users" group on the remote system. The computer is then ready for remote debugging.

5.2 Accessing the Application Using Your Own Instance

If the remote computer is set up, then you can access the application using your own instance of Visual Studio .NET. The application to be debugged must be on the remote computer. If not, copy the relevant files to this computer.

The output path for the development project must correspond to the path on the remote computer. Modify the output path for the development project if required.

The files currently in this path must be transmitted 1:1 to the remote computer. It may be useful to enable the directory on the remote computer.

To debug an application, proceed as follows within Visual Studio .NET IDE:

- Open the project file for the application.
- Access the properties of the application via the "Project/Properties" menu.
- Select the "Debug" category in the "Configuration Properties" folder of the Properties window.
- Set "Enable Remote Debugging" to "true".
- For the remote computer setting, enter the computer name or the IP address of the remote computer.
- If debugging is to be executed in mixed mode (managed and unmanaged), set "Enable Unmanaged Debugging" to "true".
- Ensure that the output path under "Configuration Properties/Create/Outputs" corresponds to the path on the remote computer.
- Click "OK" to save the changes.

You can now start debugging the application.

- From the "Debug" file menu, select "Start" to start the application on the remote computer.

You can insert breakpoints in the code within Visual Studio .NET, at which the remote program will interrupt execution. The code can then be executed in steps (or another debugging method used) in order to isolate any possible runtime problems.

The same approach also works for other .NET programming languages such as VB.NET.

5.3 Possible Problems

The Remote Debugger is an excellent tool in Visual Studio .NET IDE, however, it can still cause problems in practice. It may be impossible to receive administrator rights on the remote computer. System administrators become very nervous if they are asked to give someone administrator rights on their own computer, and are similarly reluctant when it comes to installing new applications on the computer. This can cause a problem, above all in a production environment.

5.4 Alternative Methods

If you cannot work with the Remote Debugger, e.g., because you do not have access rights for the remote computer, you must choose alternative methods.

One alternative to debugging or monitoring code in a production application is to record runtime errors in the event log or in a corresponding database. These messages can also be sent by e-mail.

Another option is to use the "Exception Handling Application Block" and the "Logging and Instrumentation Application Block". Both products are available from Microsoft free of charge.