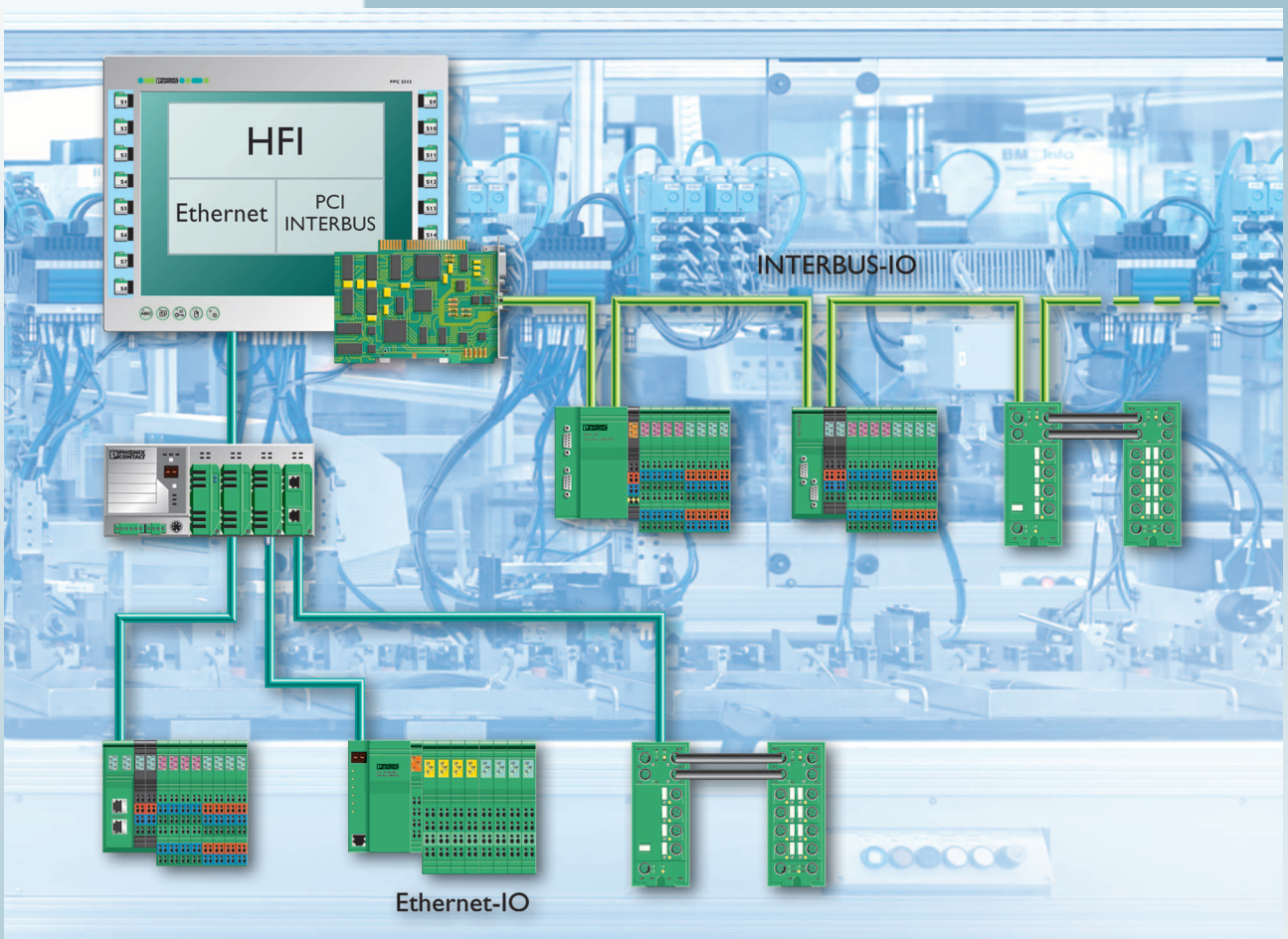


Quick start guide

UM QS EN HFI PROG

Order No.: —

Programming in high-level language
using the HFI user interface

PHOENIX CONTACT
INSPIRING INNOVATIONS

Quick start guide

Programming in high-level language using the HFI user interface

2011-11-17

Designation: UM QS EN HFI PROG

Revision: 02

Order No.: —

This user manual is valid for:

Designation	Version
HFI	3.0x

Please observe the following notes

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.

Explanation of symbols used and signal words

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety measures that follow this symbol to avoid possible injury or death.

There are three different categories of personal injury that are indicated with a signal word.

DANGER This indicates a hazardous situation which, if not avoided, will result in death or serious injury.

WARNING This indicates a hazardous situation which, if not avoided, could result in death or serious injury.

CAUTION This indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.

This symbol together with the signal word **NOTE** and the accompanying text alert the reader to a situation which may cause damage or malfunction to the device, hardware/software, or surrounding property.

This symbol and the accompanying text provide the reader with additional information or refer to detailed sources of information.

How to contact us

Internet

Up-to-date information on Phoenix Contact products and our Terms and Conditions can be found on the Internet at:
www.phoenixcontact.com

Make sure you always use the latest documentation.
It can be downloaded at:
www.phoenixcontact.net/catalog

Subsidiaries

If there are any problems that cannot be solved using the documentation, please contact your Phoenix Contact subsidiary.
Subsidiary contact information is available at www.phoenixcontact.com.

Published by

PHOENIX CONTACT GmbH & Co. KG
Flachsmarktstraße 8
32825 Blomberg
GERMANY

Should you have any suggestions or recommendations for improvement of the contents and layout of our manuals, please send your comments to:
tecdoc@phoenixcontact.com

General terms and conditions of use for technical documentation

Phoenix Contact 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 user documentation) does not constitute any further duty on the part of Phoenix Contact to furnish information on modifications to products and/or technical documentation. You are responsible to verify the suitability and intended use of the products in your specific application, in particular with regard to observing the applicable standards and regulations. All information made available in the technical data is supplied without any accompanying guarantee, whether expressly mentioned, implied or tacitly assumed.

In general, the provisions of the current standard Terms and Conditions of Phoenix Contact apply exclusively, in particular as concerns any warranty liability.

Phoenix Contact reserves the right to register its own intellectual property rights for the product identifications of Phoenix Contact products that are used here. Registration of such intellectual property rights by third parties is prohibited.

Other product identifications may be afforded legal protection, even where they may not be indicated as such.

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. Adapt them to VB or use the HFI code generator (see Section “HFI Code Generator” on page 4-8) to create a VB projet for your bus configuration. 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 class libraries. At signal level, the HFI library supports PCI cards with direct INTERBUS master and bus couplers with Ethernet connection (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 3.0x and the development system that is compatible.

Table 1-1 System requirements for HFI

Product (setup)	Environment	Development system
HFI 3.0x	Windows XP +SP3, Windows 7 (32), Windows 7 (64)	Microsoft Visual Studio 2008, C#, VB.NET)

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

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.phoenixcontact.net/catalog 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 IL 24 BK-B-PAC FL IL 24 BK-PAC IL ETH BK DI8 DO4-2TX-PAC

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.phoenixcontact.net/catalog 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 3.5, HFI 3.0” to access example projects and HFI tools.
– The required assemblies for the libraries are located in the following directory: (..\DotNet Framework 3.5\HFI 3.0\Libraries)
– The file names for the assemblies have the extension (_FX11, _FX20, _FX35..., 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

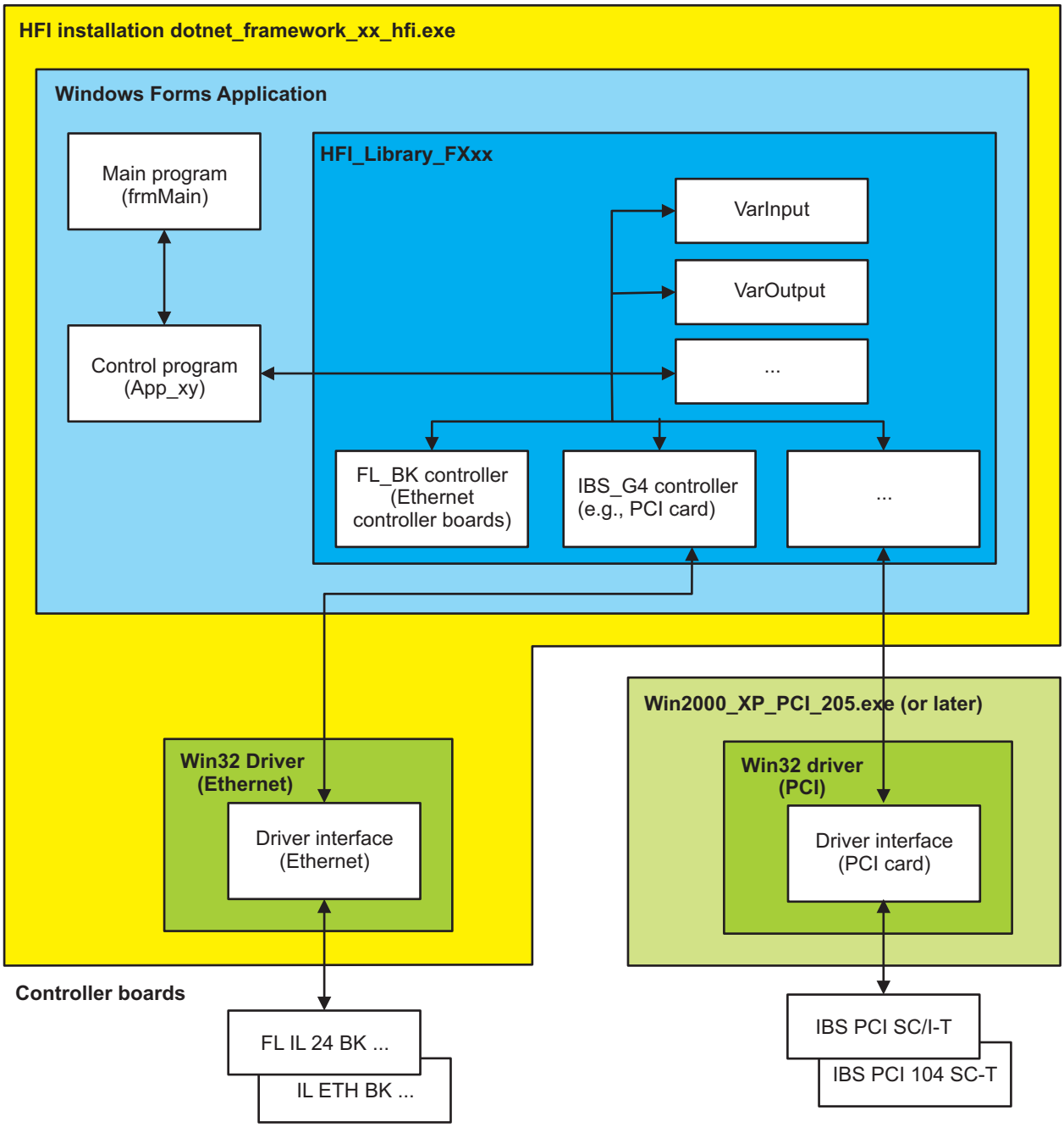


Figure 2-1 Structure of an HFI application

3 Example program in C#

The example was created using Microsoft Visual Studio 2008. 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, DotNetFramework 3.5, HFI 3.0, HFI_Demo_CS”.

In Solution Explorer, the “References” folder contains the integrated program libraries HFI_Library_FX35, PxC_Forms, PxC_Util and PxC_WinForms.

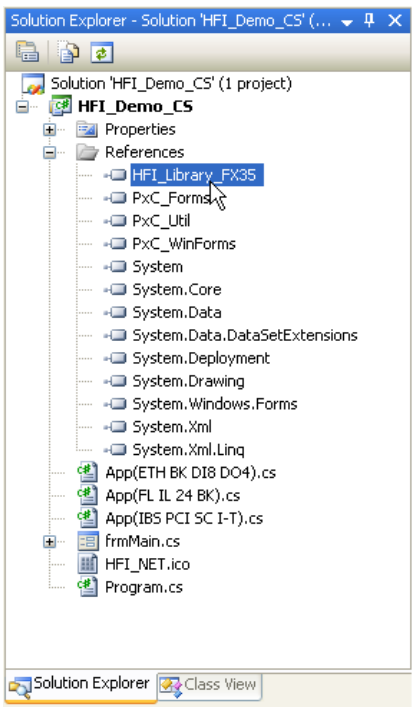


Figure 3-1 Integrated program libraries

- 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”.

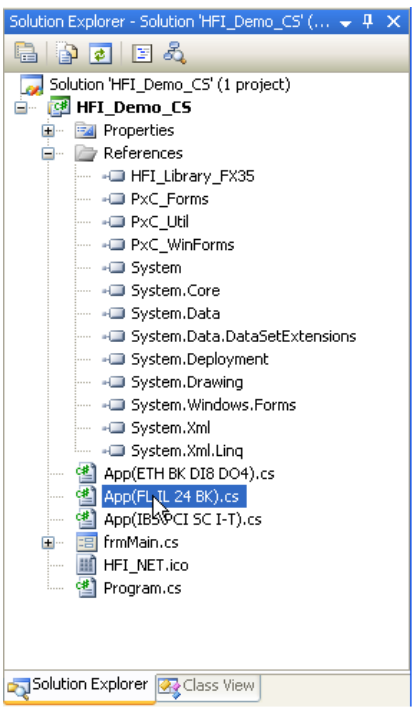


Figure 3-3 Opening the example program for the FL IL 24 BK-PAC

```
namespace HFI_Demo_CS
{
    /// <summary>
    /// This class represents an application for one controler.
    /// </summary>
    public sealed class App_FL_IL_24_BK :IDisposable
    {

        ShowLogMessage sendMessage; // Event handler

        *** Variable Declaration ***********************************************

        *** Constructor / Destructor / IDisposable Declaration *****************

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

        *** Enable / Disable the Controller ************************************

        *** 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:

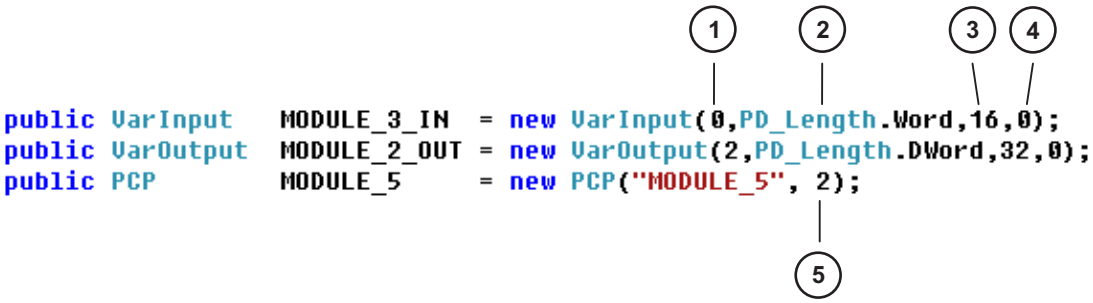


Figure 3-5 Parameters for input and output variables

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 controller
public Controller_IBS_ETH Controller;

// Create the variables for the input data
// First input terminal DI 16
public VarInput            IN_Bit_0       = new VarInput(2, PD_Length.Word, 1, 0);
public VarInput            IN_Bit_1       = new VarInput(2, PD_Length.Word, 1, 1);
public VarInput            IN_Variable    = new VarInput(2, PD_Length.Word, 12, 4);

// Second input terminal DI 32
public VarInput            IN_ByteArray   = new VarInput(4, 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(2, PD_Length.Word, 1, 0);
public VarOutput           OUT_Bit_1     = new VarOutput(2, PD_Length.Word, 1, 1);
public VarOutput           OUT_Variable  = new VarOutput(2, PD_Length.Word, 12, 4);

// Second output terminal DO 32
public VarOutput           OUT_ByteArray = new VarOutput(4, 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 Boolean            _firstStartPcp;

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 / Destructor / IDisposable Declaration *********************

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

   // Settings of the controller
   Controller.Description = "FL IL 24 BK for Demonstaration";
   Controller.Connection = "10.250.130.106";
   Controller.Startup     = ControllerStartup.PhysicalConfiguration;

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

   // The Controller Configuration property contains special configurations for the controller
   //Controller.Configuration.DNS_NameResolution     = true;
   //Controller.Configuration.EnableIBS_Indications  = true;
   Controller.Configuration.ExpertModeActivate     = false;
   //Controller.Configuration.GetRevisionInfo        = false;
   //Controller.Configuration.Read_IBS_Cycletime     = 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. (Exception: The IBS PCI SC/I-T and IBS PCI 104 SC-T controller boards do not require an “Expert Mode”.

NOTE:
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.

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.OnException          += new ExceptionHandler(Controller_OnException);


// Events from PCP_2
PCP_RS232_1.OnEnableReady               +=new EnableReadyHandler(PCP_RS232_1_OnEnableReady);
PCP_RS232_1.OnReadConfirmationReceived  +=new ReadConfirmationReceiveHandler(PCP_RS232_1_ReadConfirmationReceived);
PCP_RS232_1.OnWriteConfirmationReceived +=new WriteConfirmationReceiveHandler(PCP_RS232_1_WriteConfirmationReceived);
PCP_RS232_1.OnException                 += new ExceptionHandler(PCP_RS232_1_OnException);
```

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.

NOTE:
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 “Controller_OnUpdateProcessData” 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 Controller_OnUpdateProcessData

OnUpdateMailbox

The “OnUpdateMailbox” event is called cyclically at the interval set for the mailbox update time (50 ms).
In the “Controller_OnUpdateMailbox” 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.BusDiag.StatusRegister.RUN)
   {
      if (!PCP_RS232_1.Ready && !PCP_RS232_1.Error)
      {
         if(!_firstStartPcp)
         {
            _firstStartPcp = true;
            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 Controller_OnUpdateMailbox

OnException (Controller)

The “OnException” event is called on a change in the diagnostic status of the “Controller” class.
The “Controller_OnException” function registered in the “OnException” event (see Figure 3-11 on page 3-7) displays the current diagnostic message in a text box.

```
/// <summary>
///  Called whenever an error occurs in the controller object
/// </summary>
/// <param name="Sender"></param>
/// <param name="Diagnostic"></param>
void Controller_OnException(Exception ExceptionData)
{
   if (sendMessage != null)
   {
      sendMessage(Diagnostic.GetExceptionMessage(ExceptionData));
   }

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

Figure 3-14 Controller_OnException

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[] Dat
{
   // 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.

```
/// <summary>
/// Called for each successfull write confirmation
/// </summary>
/// <param name="Sender"></param>
/// <param name="Data"></param>
private void PCP_RS232_1_WriteConfirmationReceived(object Sender)
{
    // TODO insert your code here
}
```

Figure 3-16 PCP_RS232_1_WriteConfirmationReceived

OnException (PCP)

The “OnException” event is called when an error occured with PCP communication.
The “PCP_RS232_1_OnException” function registered in the “OnException” event (see Figure 3-11 on page 3-7) displays the current diagnostic message in a text box.

```
/// <summary>
/// Called whenever an error occurs in the pcp object
/// </summary>
/// <param name="Sender"></param>
/// <param name="Diagnostic"></param>
void PCP_RS232_1_OnException(Exception ExceptionData)
{
   if (sendMessage != null)
   {
      sendMessage(Diagnostic.GetExceptionMessage(ExceptionData));
   }

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

Figure 3-17 PCP_RS232_1_OnException

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 Controller ************************

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

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();
}
```

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 (see Figure 3-22).

```
#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()
{
   if (Controller != null)
   {
      if (Controller.Connect || Controller.Error)
      {
         Controller.Disable();

         while (Controller.Connect || Controller.Error)
         {
            System.Threading.Thread.Sleep(10);
         }
      }

      Controller.Dispose();
   }
}

#endregion

}

}
```

Figure 3-21 Exiting the program

3.7 Function for data exchange (Update the data of 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 "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 *** Update the Data of the Form ***************************************

/// <summary>
/// Update the main form
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void tmrMainFormUpdate_Tick(object sender, EventArgs e)
{
   // Show the controller state
   cbxConnect.Checked   = myApplication.Controller.Connect;
   cbxRun.Checked       = myApplication.Controller.Run;
   cbxError.Checked     = myApplication.Controller.Error;
   cbxWatchdog.Checked  = myApplication.Controller.WatchdogOccurred;

   // Show the INTERBUS state
   cbxBusReady.Checked  = myApplication.Controller.BusDiag.StatusRegister.READY;

   ...
}
```

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”.

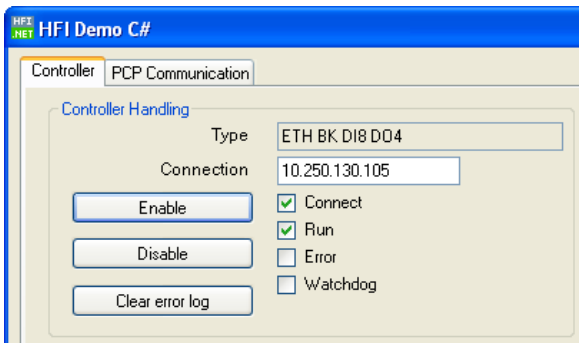


Figure 3-23 Setting the IP address and activating the “Controller” class

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

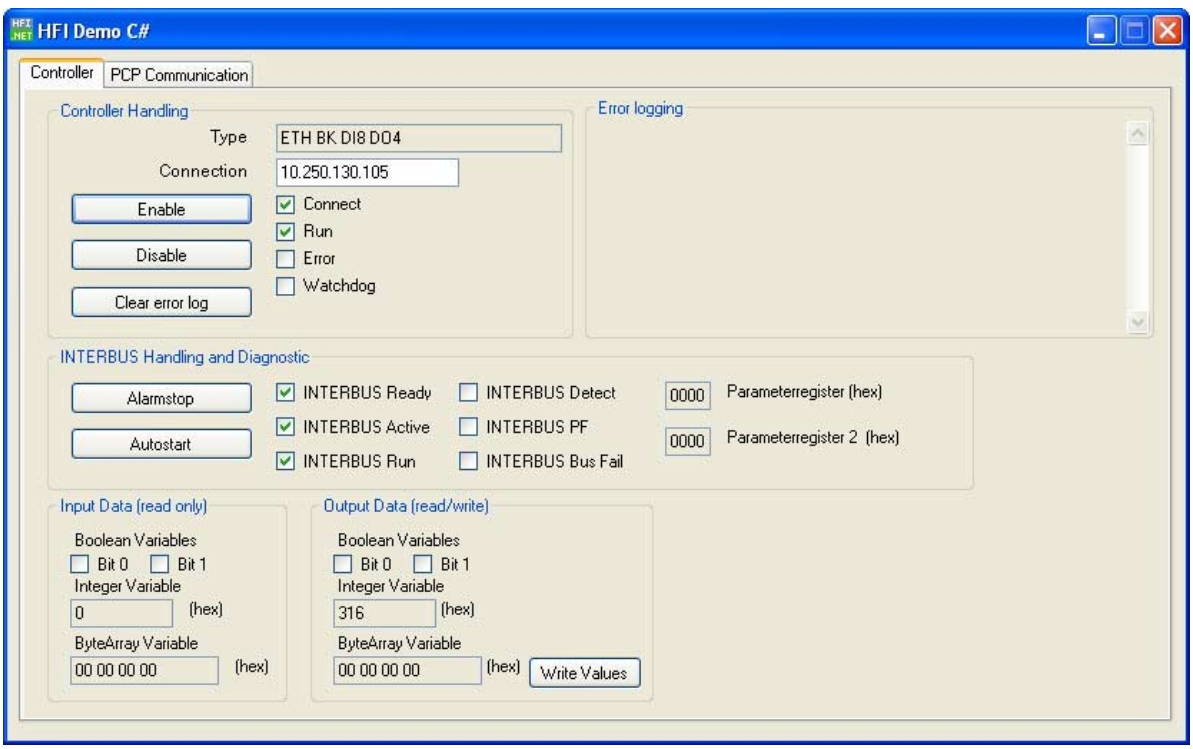


Figure 3-24 User interface for the example program

In the “Controller Handling” area, the “Run” checkbox indicates that the “Controller” class has been started successfully.

The “INTERBUS Handling and Diagnostics” 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 “Controller_OnUpdateMailbox” on page 3-9).

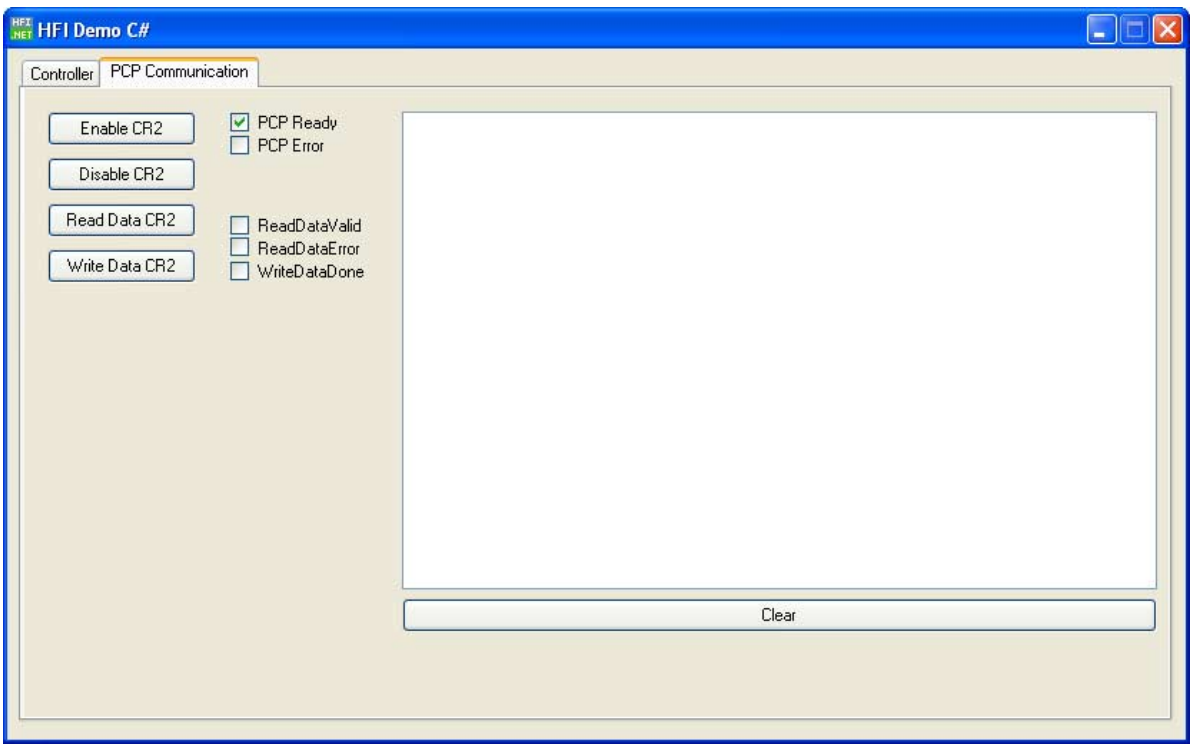


Figure 3-25 User interface for the example program: “PCP Communication” tab

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

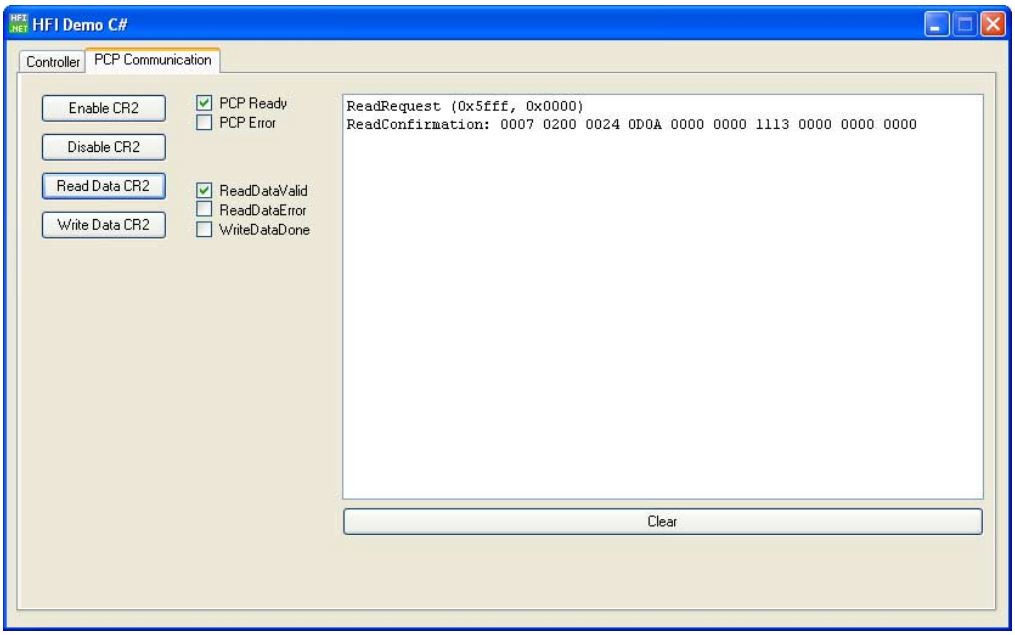


Figure 3-26 PCP data read

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

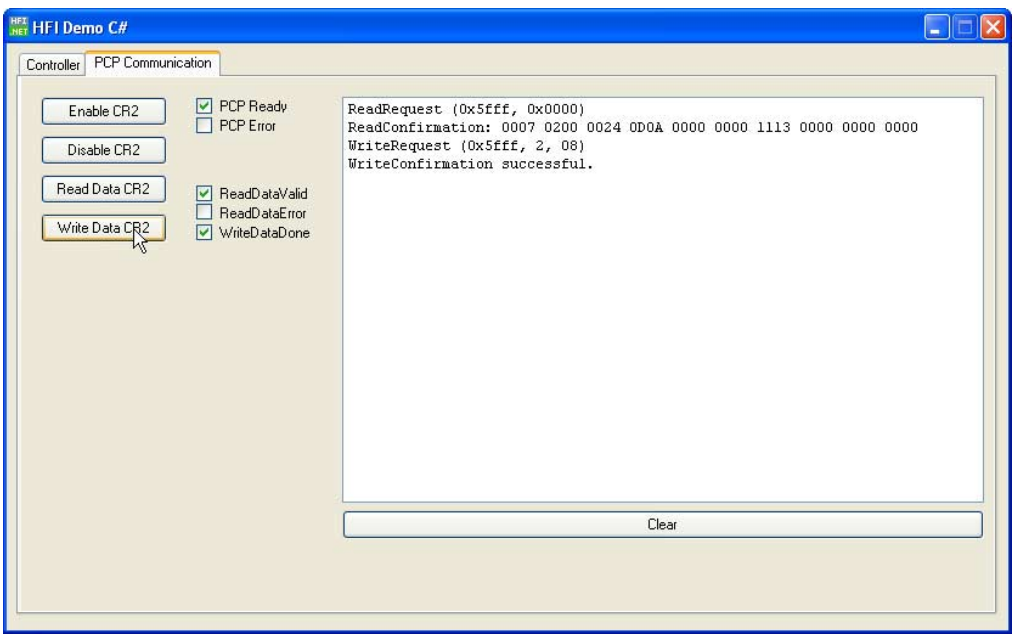


Figure 3-27 PCP data written

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

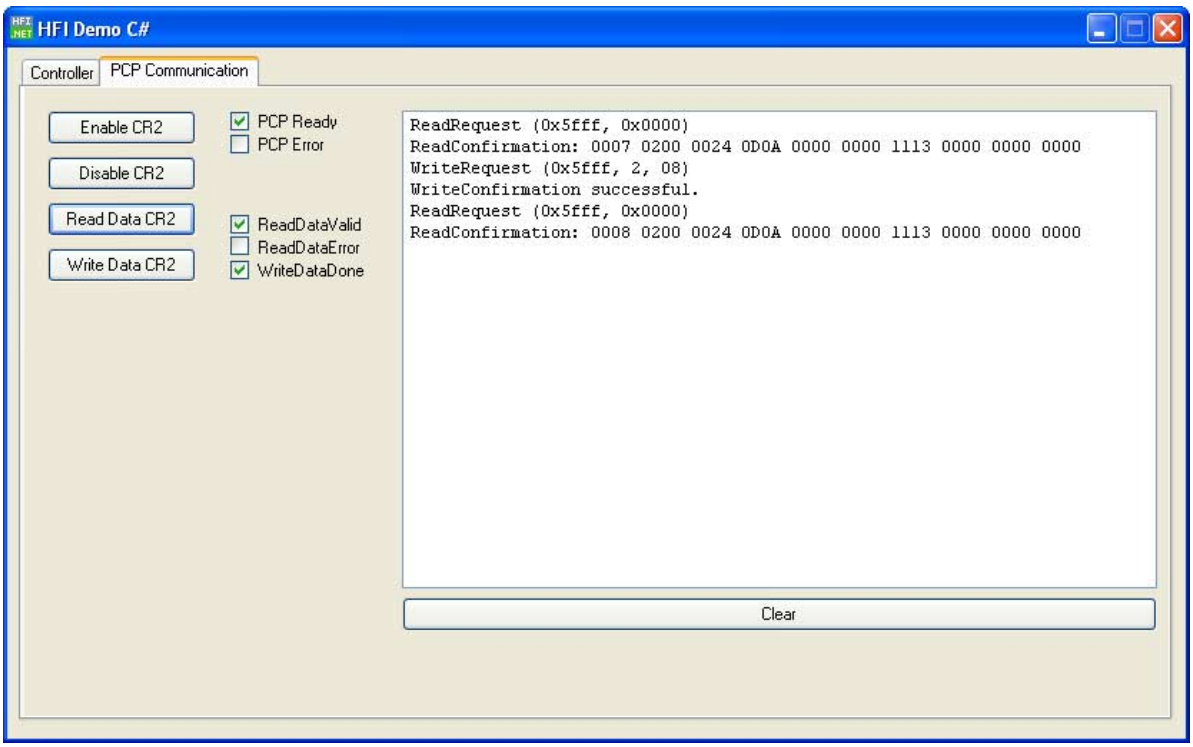


Figure 3-28 PCP data read again

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		Physical
	CMD	Plug and play	
IBS PCI SC/I-T	Yes	No	Yes
IBS PCI 104 SC-T	Yes	No	Yes
FL IL 24 BK-B-PAC	No	Yes	Yes
FL IL 24 BK-PAC	No	Yes	Yes
IL ETH BK DI8 DO4 2TX-PAC	No	Yes	Yes

Logical configuration

- 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.

Physical configuration

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 IL 24 BK-B-PAC	No	Yes
FL IL 24 BK-PAC	No	Yes
IL ETH BK DI8 DO4 2TX-PAC	No	Yes

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.
- Deactivate the process data watchdog (recommended). To do this set the value for “Process Data Watchdog Timeout” to 0 ms.

The HFI has its own watchdog.

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.

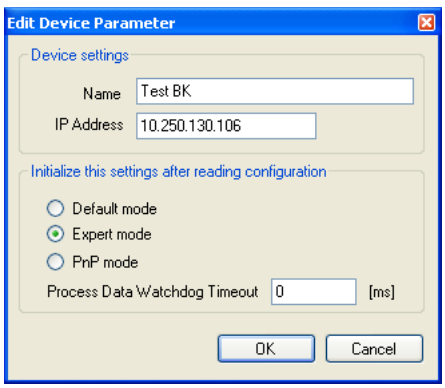


Figure 4-1 HFI Device Explorer: “Add New Device / Edit Device Parameter” window

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”.

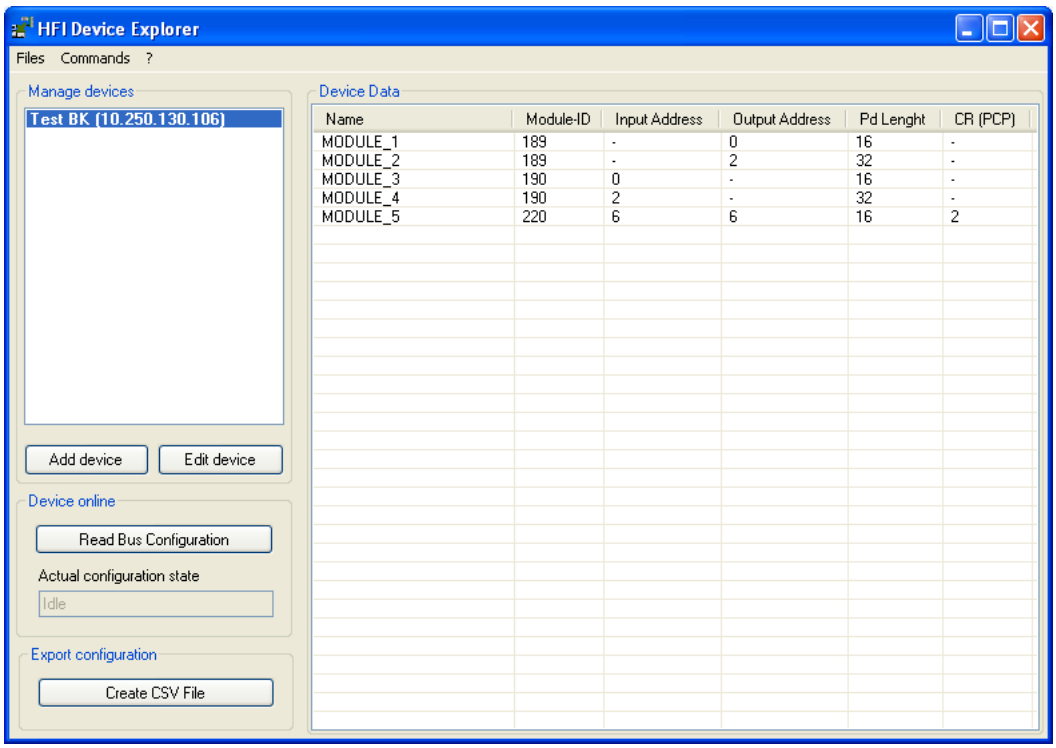


Figure 4-2 HFI Device Explorer

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

Device Data

Name	Module-ID	Input Address	Output Address	Pd Lenght	CR (PCP)
MODULE_1	189	-	0	16	-
MODULE_2	189	-	2	32	-
MODULE_3	190	0	-	16	-
MODULE_4	190	2	-	32	-
MODULE_5	220	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

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.

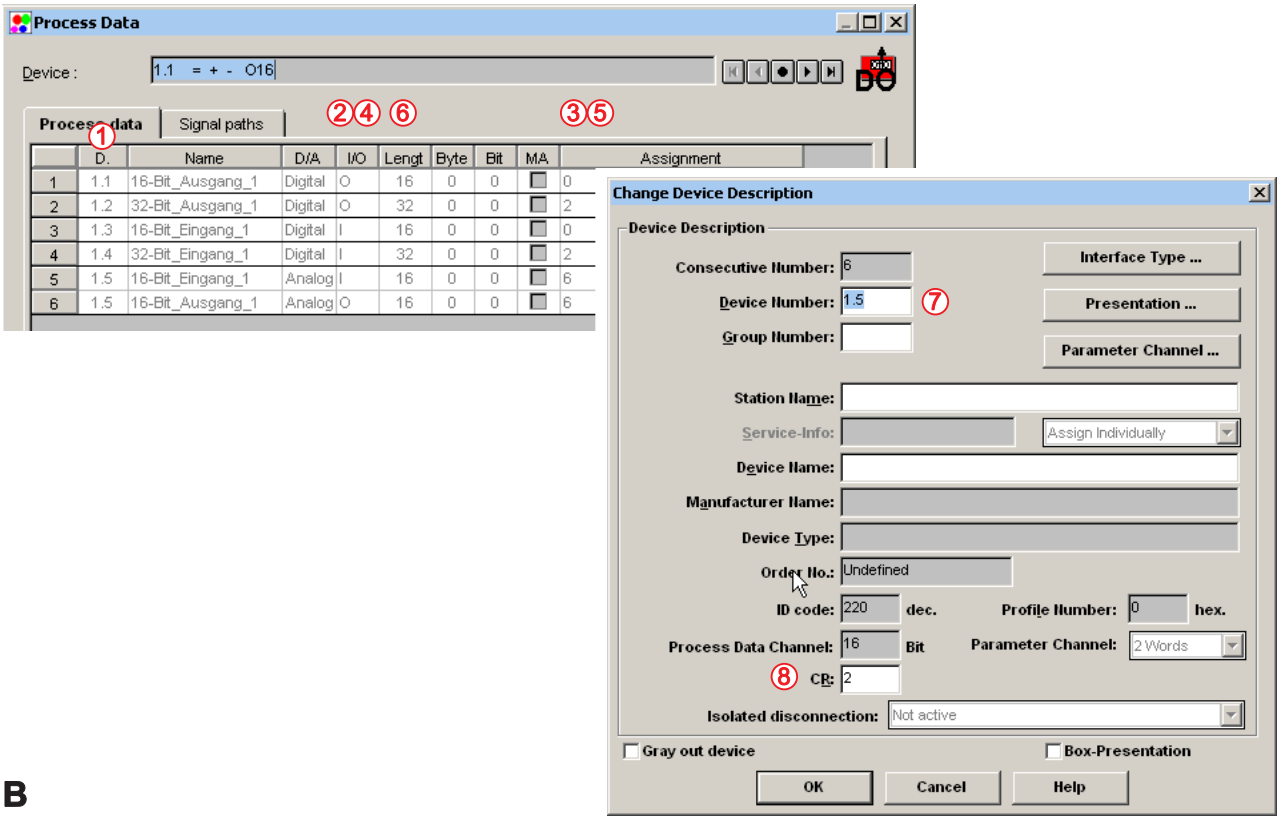
Proceed as follows:

- 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.

- Click on “Read CSV File”.
  After you have opened the CSV file, the ceck boxes show the status of the analyzed data.

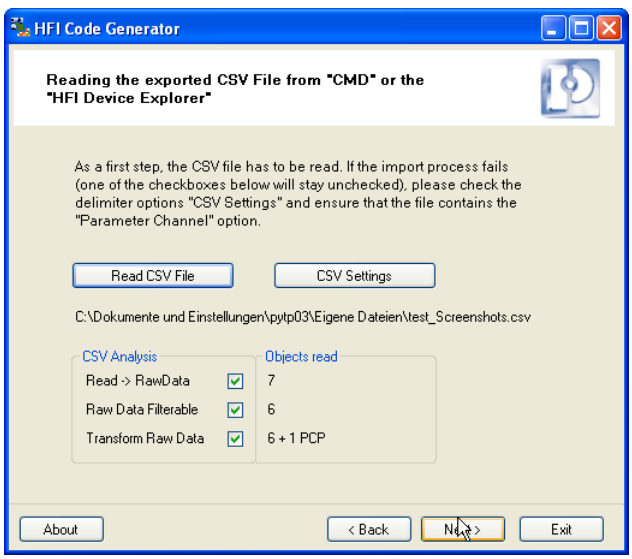


Figure 4-6 Read CSV file

- Select the template (e.g., “VS2008 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”.

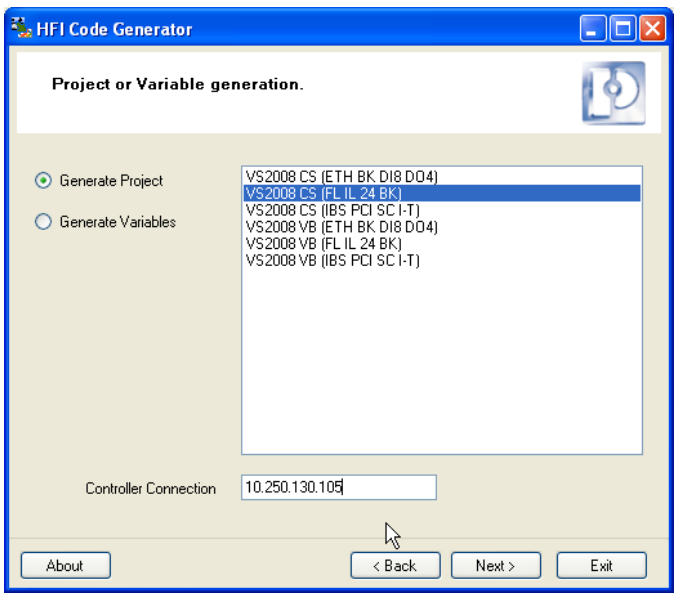


Figure 4-7 Template and IP address for the FL IL 24 BK-PAC

- In the window that opens, select the path for the project to be created.

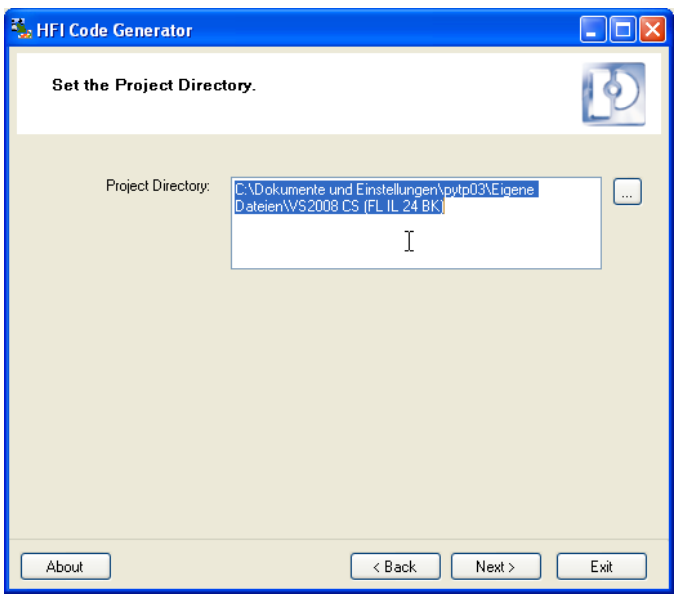


Figure 4-8 Select path

- In the window that opens, click on “Next” 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.

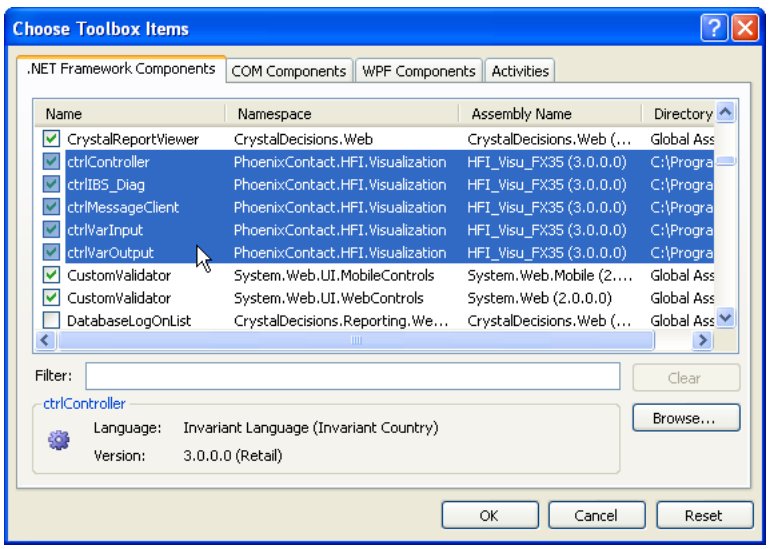


Figure 4-10 HFI controls

4.6.2 Functions of the controls

ctrlController

Read and operate the controller

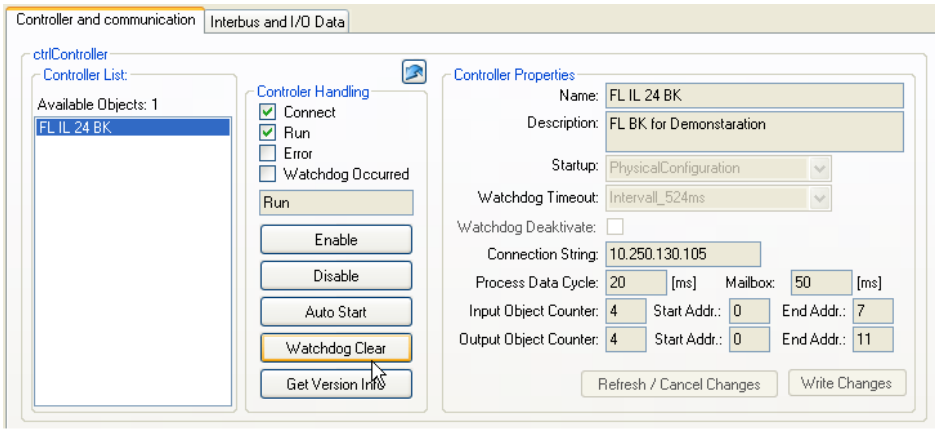


Figure 4-11 Controls: ctrlController

ctrlIBS_Diag

INTERBUS diagnostics and bus handling

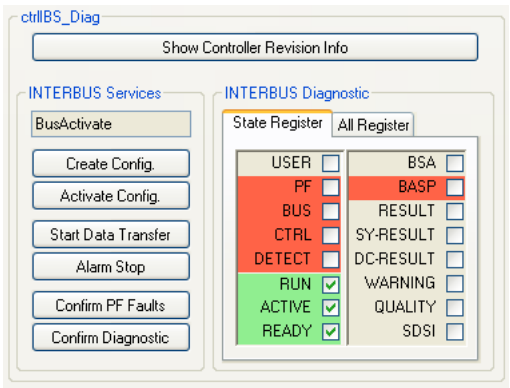


Figure 4-12 Controls: ctrlIBS_Diag

ctrlMessageClient Read PCP and firmware telegrams in the active application

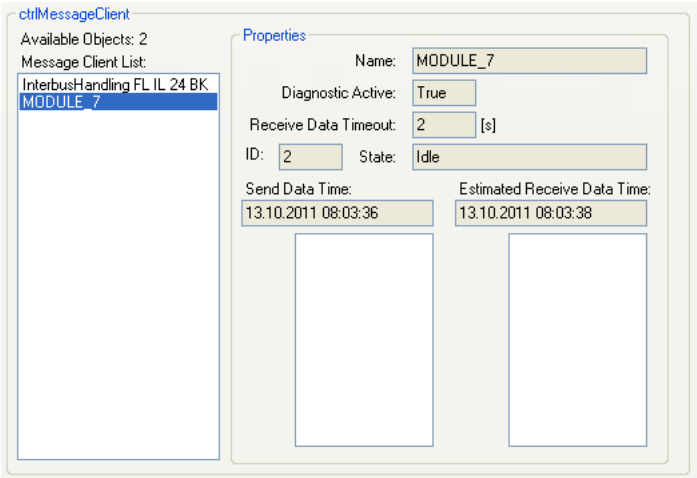


Figure 4-13 Controls: ctrlMessageClient

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

ctrlVarOutput Read and write the properties of an output object

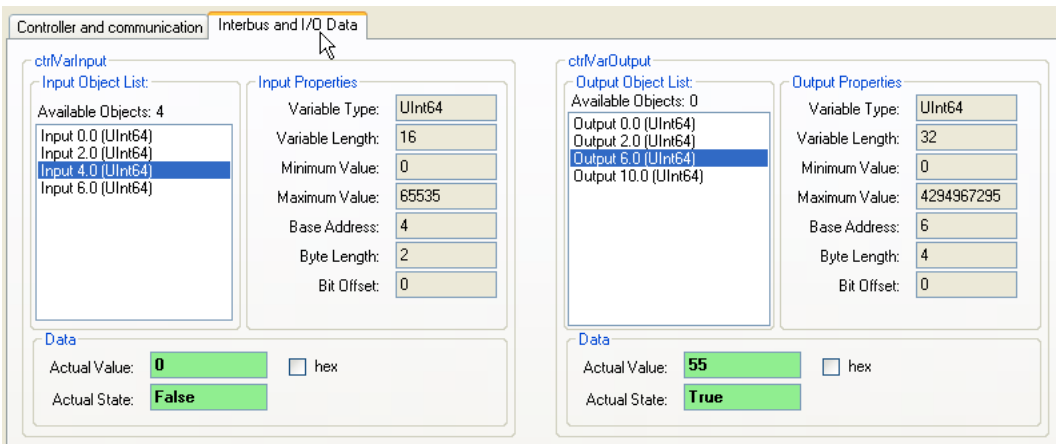


Figure 4-14 Controls: ctrlVarInput, ctrlVarOutput

When the “EditActivate” (in the source code) property is set, the “Actual Value:” output variable value of the selected output object can be edited.

SCATTERGOOD & JOHNSON LTD
ELECTRICAL ENGINEERING & FLUID CONTROL DISTRIBUTORS
Est.1899

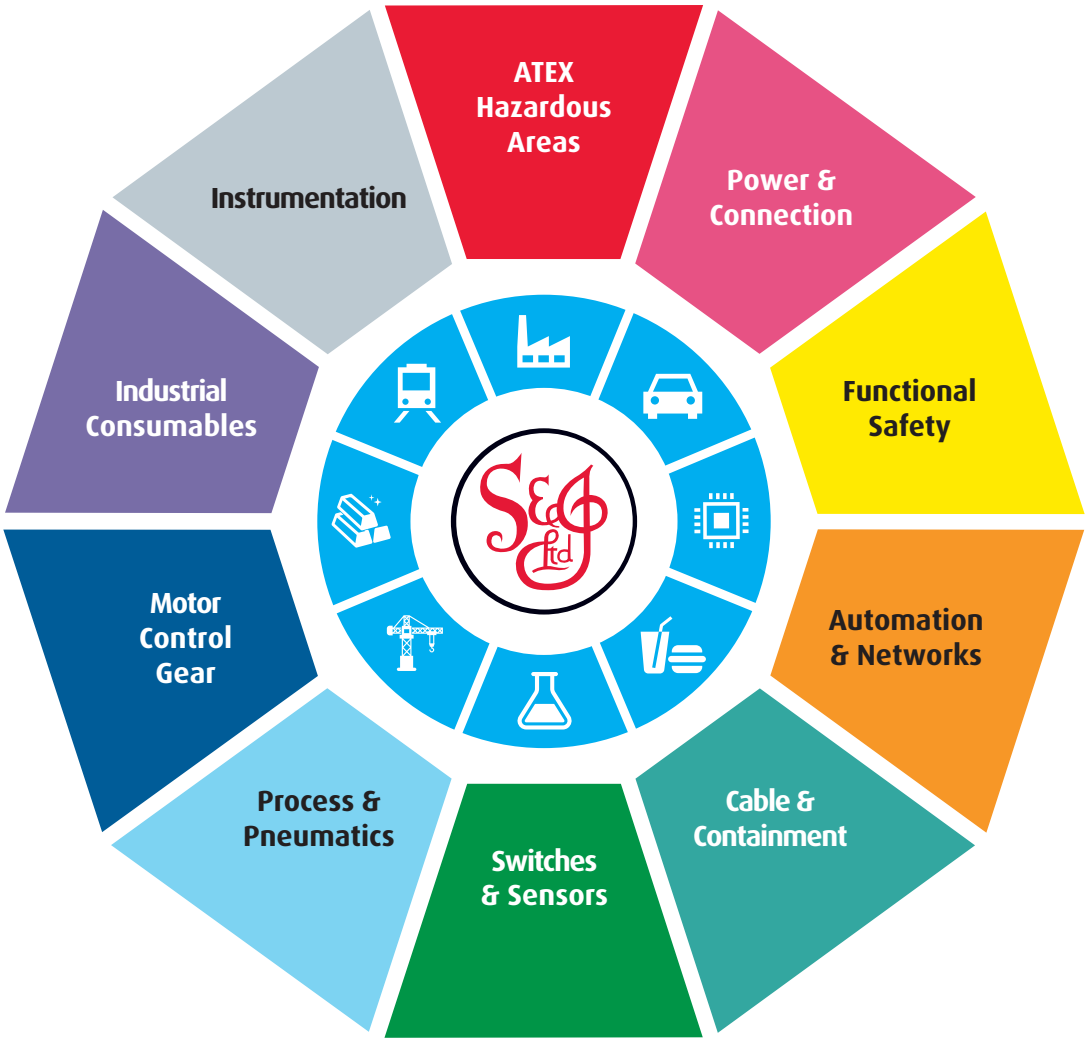
At Scattergood & Johnson Ltd, we pride ourselves on being a technical distributor to specialist industries.

Working with a range of quality product suppliers across a number of specialist markets, we are not your average 'box shifter' - we are your technical and supply chain partner.

We fully support every product we sell - for free! Our internal team and external sales engineers can answer any product or application question, no matter the complexity.

Backing up this technical ability is a range of 50,000+ products available from stock for nationwide next day delivery (same day if required!), or you can collect what you need from any of our trade counters around the UK.

Select your specialist interest below to learn more about how we can help.



Online, In Branch and On the Road - Scattergood & Johnson Ltd, there when you need us.

www.scatts.co.uk