SIEMENS

SIMATIC

WinAC ODK V4.1
Controller Management Interface

User Manual

Version: 12 Apr 2004

A5E00340324-01

Copyright and Safety Notification

This manual contains notices that you should observe to ensure your own personal safety, as well as to protect the product and connected equipment. These notices are highlighted in the manual by a warning triangle and are marked as follows according to the level of danger:

Danger
Indicates an imminently hazardous situation that, if not avoided, will result in death or serious injury.

Warning
Indicates a potentially hazardous situation that, if not avoided, could result in death or severe injury.

Caution
Used with the safety alert symbol indicates a potentially hazardous situation that, if not avoided, may result in minor or moderate injury.

Caution
Used without the safety alert symbol indicates a potentially hazardous situation that, if not avoided, may result in property damage.

Notice
NOTICE used without the safety alert symbol indicates a potential situation that, if not avoided, may result in an undesirable result or state.

Qualified Personnel

The device/system may only be set up and operated in conjunction with this manual. Only qualified personnel should be allowed to install and work on this equipment. Qualified persons are defined as persons who are authorized to commission, to ground, and to tag circuits, equipment, and systems in accordance with established safety practices and standards.

Correct Usage

Note the following:

Caution
This device and its components may only be used for the applications described in the catalog or the technical descriptions and only in connection with devices or components from other manufacturers that have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored, set up, and installed correctly, and operated and maintained as recommended.

Trademarks

Siemens® and SIMATIC® are registered trademarks of SIEMENS AG.
Microsoft ®, Windows ®, Windows XP Professional ®, Windows 2000 ®, and Internet Explorer ® are registered trademarks of Microsoft Corporation.
RTX™ is a trademark of Venturcom, Inc.

Copyright Siemens Energy & Automation, Inc., 2004
All rights reserved
The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

Siemens AG
Bereich Automatisierungs- und Antriebstechnik
Geschaeftsgebiet Industrie-Automatisierungssysteme
Postfach 4848, D-90327 Nuernberg

Disclaimer of Liability
We have checked the contents of this manual for agreement with the hardware and software described. Because deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcomed.

© Siemens AG 2004

Technical data subject to change.

Preface

The WinAC Controller Management Interface (CMI) is a component of WinAC ODK V4.1. With CMI, you can develop a custom software application that can operate any of the WinAC controllers. For example, your custom application can implement a control panel that displays controller status information and controls the operating mode of the controller. You can also use CMI to access additional controller data. CMI supports the following controllers:

- WinLC Basis V4.1
- WinLC RTX V4.1
- WinAC Slot PLC (CPU 412-2 PCI and CPU 416-2 PCI)

This online user manual is included with the WinAC ODK V4.1 installation.

Audience

This manual is intended for engineers and programmers who have software design and programming experience as well as a general knowledge of programmable logic controllers. The WinAC Controller Management Interface is an open interface, supporting the following programming environments:

- Visual Basic 6.0
- Visual Basic .NET
- Visual C++ 6.0
- Visual C++ .NET
- Visual C# .NET
- Borland Delphi 7

Scope

This document describes the features and the operation of WinAC Controller Management Interface version 4.1.

Note
The online help version of this documentation provides popup windows with highlighted examples from code listings to illustrate various programming tasks. These popups of highlighted code listings are not available from the PDF version of this documentation. The Reference section provides complete code listings for the example programs in four programming environments.

Other Manuals

For additional information about your controller, refer to the following documentation:

Title	Content
Windows Automation Center WinAC Basis V4.1 User Manual	This manual provides information about WinAC Basis and the WinLC Basis controller for Windows 2000 or Windows XP
Windows Automation Center RTX WinAC RTX 4.1 User Manual	This manual provides information about WinAC RTX and the WinLC RTX controller for Windows 2000 or Windows XP with the Venturcom Real-time extensions (RTX).
Windows Automation Center WinAC Slot 412/WinAC Slot 416 Version 3.4 Overview Manual	This manual provides information about the functionality of WinAC Slot 412 and WinAC Slot 416.
WinAC Controlling with CPU 412-2 PCI/CPU 416-2 PCI Setting Up, CPU Data Version 3.4	This manual provides installation and configuration information, as well as technical specifications, for the CPU 412-2 PCI and the CPU 416-2 PCI.
Windows Automation Center Open Development Kit (WinAC ODK) V4.1 User Manual	This manual provides information about the open development kit (ODK) software package for WinAC controllers. The Controller Management Interface (CMI) is a component of ODK.

Additional Assistance

For assistance in answering technical questions, for training on this product, or for ordering, contact your Siemens distributor or sales office.

To contact Customer Service for Siemens in North America and in South America:

- Telephone: +1 (800) 333-7421
- Fax: +1 (423) 262-2200
- Email: simatic.hotline@sea.siemens.com

To contact Customer Service for Siemens in Europe and in Africa:

- Telephone: +49 (0) 180 5050 222
- Fax: +49 (0) 180 5050 223
- Email: adsupport@siemens.com

To contact Customer Service for Siemens in Asia and in the Pacific region:

- Telephone: +86 10 64 75 75 75
- Fax: +86 10 64 74 74 74
- Email: adsupport.asia@siemens.com

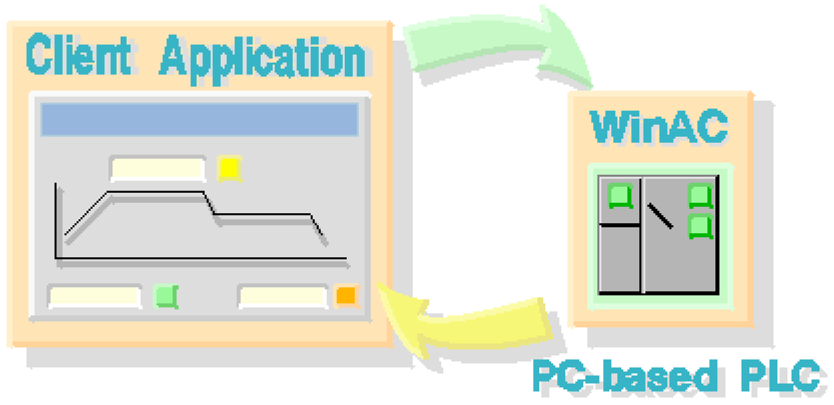
Contents

Product Overview

The WinAC Controller Management Interface (CMI) allows you to create a custom client application that interacts directly with a PC-based PLC, such as WinLC Basis, WinLC RTX, or a slot PLC. Your application runs as a client of the PLC and interacts with the PLC by reading or writing specific features of the PLC that are defined in the interface.

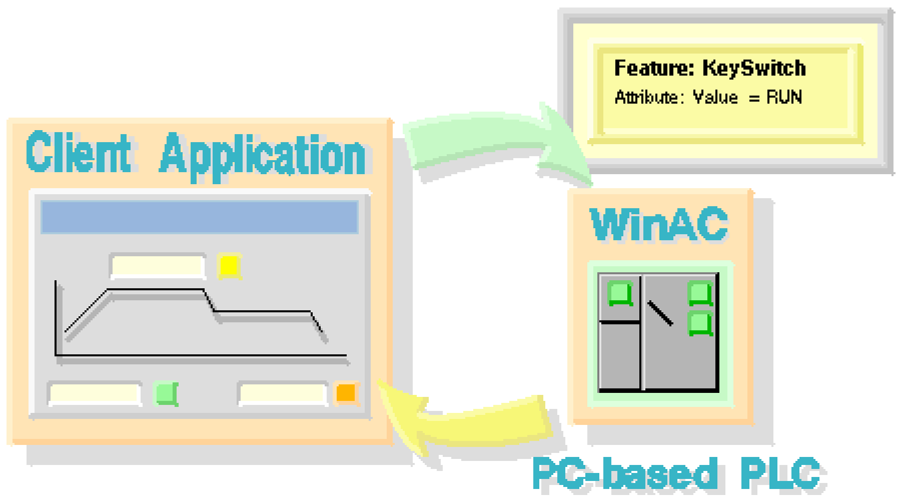
Your custom application can implement all of the functionality of the WinAC control panel using the CMI interface. It can display status indicators, change the operating mode, display and change tuning data, read the diagnostic buffer, and perform other functions using the published features of the Controller Management Interface.

The open nature of the WinAC Controller Management Interface allows you to develop applications in any of the programming environments supported by CMI: Visual Basic 6.0, Visual Basic .NET, Visual C++ 6.0, Visual C++ .NET, Visual C# .NET, or Borland Delphi 7.



The Controller Management Interface provides a set of features for WinAC controllers. Each feature includes a set of attributes that allow your custom application to monitor or modify specific aspects of controller operation.

For example, the feature KeySwitch provides a Value attribute with specific settings (STOP, RUN, RUNP, or MRES) that correspond to the positions of the keyswitch (or mode buttons) of the PLC. Changing the setting of the Value attribute of the KeySwitch feature is like changing the position of the keyswitch of the PLC.

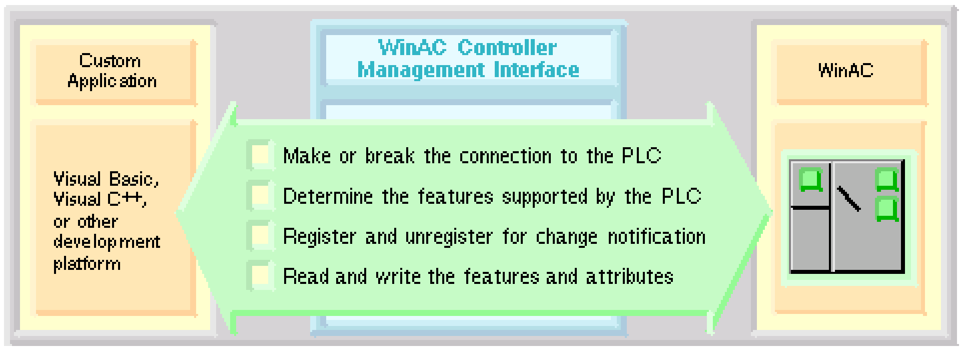


Capabilities of the WinAC Controller Management Interface

The WinAC Controller Management Interface provides a FeatureProvider COM object. The Feature Provider enables your application to perform the following functions:

- Find and connect to a PLC that is either running or is installed on your computer
- Determine the set of features (or properties) that are supported by the connected PLC. The CMI interface supports WinLC Basis, WinLC RTX, and Slot PLCs, each of which has a unique set of features.
- Read the values for attributes (or parameters) of the features supported by the PLC and CMI. Your application can use these attributes as necessary, such as displaying the LED indicators of the PLC.
- Change the values of the attributes for specific features supported by the PLC and CMI. This allows your application to perform certain functions, such as changing the operating mode of the PLC.
- Register a feature for change notification so that your application can respond to specific events in the PLC. For example, your application can detect a change to STOP mode and take specific action in that event.

The picture below shows the capabilities of the WinAC Controller Management Interface:


Custom Application
Visual Basic, Visual C++, or other development platform
WinAC Controller Management Interface
Make or break the connection to the PLC
Determine the features supported by the PLC
Register and unregister for change notification
Read and write the features and attributes
WinAC

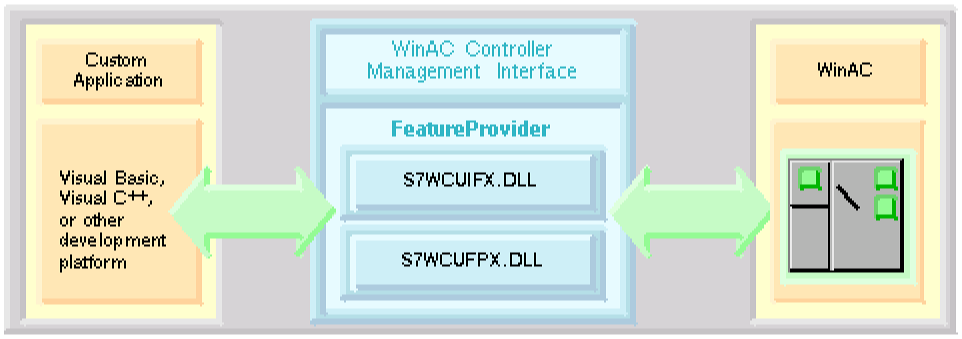
The topic "List of Features and Attributes" contains the complete set of supported features and attributes for WinLC Basis, WinLC RTX, and the Slot PLCs.

Implementing the WinAC Controller Management Interface

Your client application connects to the FeatureProvider COM object of the WinAC Controller Management Interface, which contains two DLLs that provide interfaces for accessing the functionality of the PLC:

- S7WCUIFX.DLL is the "S7 WinAC Unified Panel Interfaces 1.0 Type Library". It provides the interface definitions.
- S7WCUFPX.DLL is the "FeatureProvider 1.0 Type Library". It provides the Feature Provider that implements the interfaces.

The installation of your controller (WinLC Basis, WinLC RTX, or Slot PLC) supplies the two CMI DLLs in the default installation directory ...\Siemens\Common\OCX. The installation of WinAC ODK installs type library files that provide access to the CMI DLLs. How you incorporate the Feature Provider into your custom application depends on your programming environment, and is described in Referencing the Controller Management Interface Type Libraries.

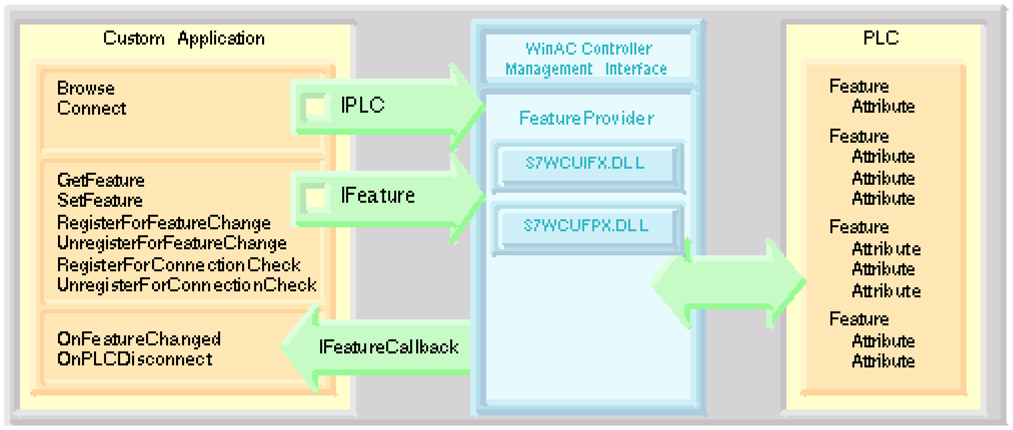

Custom Application
Visual Basic, Visual C++, or other development platform
WinAC Controller Management Interface
FeatureProvider
S7WCUIFX.DLL
S7WCUFPX.DLL
WinAC

Interfaces of the Feature Provider

The DLLs of the Feature Provider supply interfaces that your client application uses to interact with the PLC that is either running or installed on your local computer. These interfaces provide functions, or methods, for reading or writing the values of the features supported by the PLC.

Each PLC supports a set of features for monitoring or modifying operation of the connected PLC, such as a keyswitch or status LEDs. Each feature has a set of attributes that contain a value. For example, the mode switch that sets the operating mode of a PC-based PLC is controlled by the Value attribute of the KeySwitch feature.

The picture below illustrates the interfaces between the custom application and the features and attributes of the PLC. You can click areas of the picture to access detailed information about each interface or about the PLC features and attributes.



You use the methods of the following interfaces to interact with the PLC by reading and writing values for the attributes of the features:

- The IPLC interface provides methods for finding and connecting to a PLC on the local computer.
- The IFeature interface provides methods for reading or writing to an individual feature and its attributes. It provides methods that register a feature for notification of a change in the value of an attribute of that feature. In addition, it provides methods for verifying the connection to the PLC.
- The IFeatureCallback interface provides a method that notifies your client application of a change in the value of an attribute of a registered feature. It also provides a method that you can use to verify the connection to the PLC.

Note: The IFeatureCallback interface defines two methods: OnFeatureChanged and OnPLCDisconnect. You must write the code in your application to implement these methods for handling the callbacks (OnFeatureChanged) and the notification of an abnormal loss-of-connection (OnPLCDisconnect).

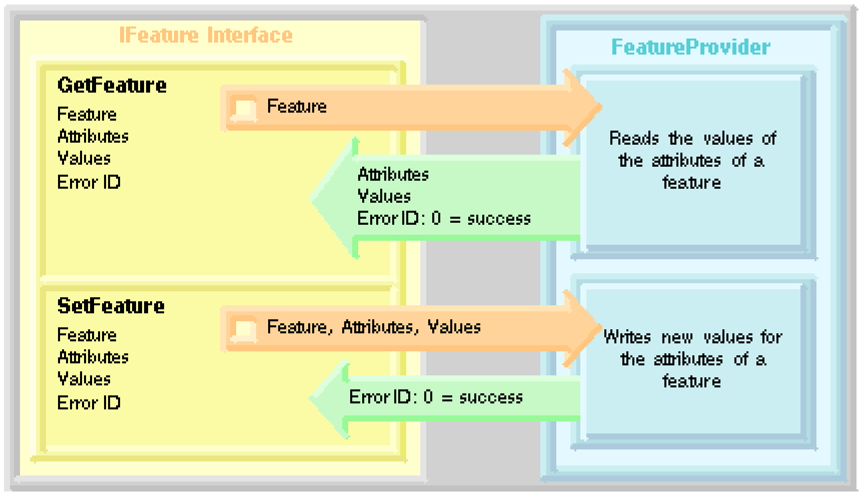
Methods of the IFeature Interface

The IFeature interface provides three sets of paired methods (six individual methods) for interacting with the PLC:

- GetFeature and SetFeature: These methods allow you to read and write values for the attributes of specific features.
- RegisterFeatureForChange and UnregisterFeatureForChange: These methods allow you to register (and to unregister) for notification (callback) whenever a readable value of a feature changes. (Not all features can be registered. Refer to the list of features to determine whether a feature can be registered for callback.)
- RegisterForConnectionCheck and UnregisterForConnectionCheck: These methods allow you to register (and to unregister) for a connection check by the Feature Provider.

The RegisterFeatureForChange and RegisterForConnectionCheck methods enable the Feature Provider to initiate the methods of the IFeatureCallback interface.

You use the GetFeature and SetFeature methods to read or write values for the attributes associated with a feature.



GetFeature

The GetFeature method reads a feature in the PLC and returns the values of the attributes.

GetFeature (FeatureName, pAttributeNames, pAttributeValues, pErrorID)

Visual Basic

```
Sub GetFeature(
   FeatureName As String,
   pAttributeNames,
   pAttributeValues,
   pErrorID As Long)
```

Visual C++

```
HRESULT GetFeature(
   [in] BSTR FeatureName,
   [in,out] VARIANT* pAttributeNames,
   [in,out] VARIANT* pAttributeValues,
   [in,out] long* pErrorID);
```

C#

```
instance void  GetFeature(
   [in] string  marshal( bstr) FeatureName,
   [in][out] object&  marshal( struct) pAttributeNames,
   [in][out] object&  marshal( struct) pAttributeValues,
   [in][out] int32& pErrorID) runtime managed internalcall
```

Borland Delphi 7

```
procedure GetFeature(
   const FeatureName: WideString;
   var pAttributeNames: OleVariant;
   var pAttributeValues: OleVariant;
   var pErrorID: Integer); safecall;
```

where:

- FeatureName defines the feature to be read by this operation.
- pAttributeNames is a pointer to an array of names for the attributes associated with the selected feature.
- pAttributeValues is a pointer to an array of values for the attributes of the selected feature..
- pErrorID is a pointer to the error number. If the error number is not 0, the operation was not successful. A value of 0 for pErrorID does not always indicate success. See Programming Tips and Error Handling.

SetFeature

The SetFeature method writes new values for the attributes of a specified feature in the PLC. A CMI application can only use SetFeature to write attributes of type Write or Read/Write.

> **Note**
> Do not pass attributes of type Read to SetFeature. SetFeature ignores any attributes that are read-only. It returns no error code and does not generate a value change in the Error feature.

SetFeature (FeatureName, AttributeNames, AttributeValues, pErrorID)

Visual Basic

```
Sub SetFeature(
   FeatureName As String,
   AttributeNames,
   AttributeValues,
   pErrorID As Long
```

Visual C++

```
HRESULT SetFeature(
   [in] BSTR FeatureName,
   [in] VARIANT AttributeNames,
   [in] VARIANT AttributeValues,
   [in,out] long* pErrorID);
```

C#

```
instance void  SetFeature(
   [in] string  marshal( bstr) FeatureName,
   [in] object  marshal( struct) AttributeNames,
   [in] object  marshal( struct) AttributeValues,
   [in][out] int32& pErrorID) runtime managed internalcall
```

Borland Delphi 7

```
procedure SetFeature(
   const FeatureName: WideString;
   AttributeNames: OleVariant;
   AttributeValues: OleVariant;
   var pErrorID: Integer); safecall;
```

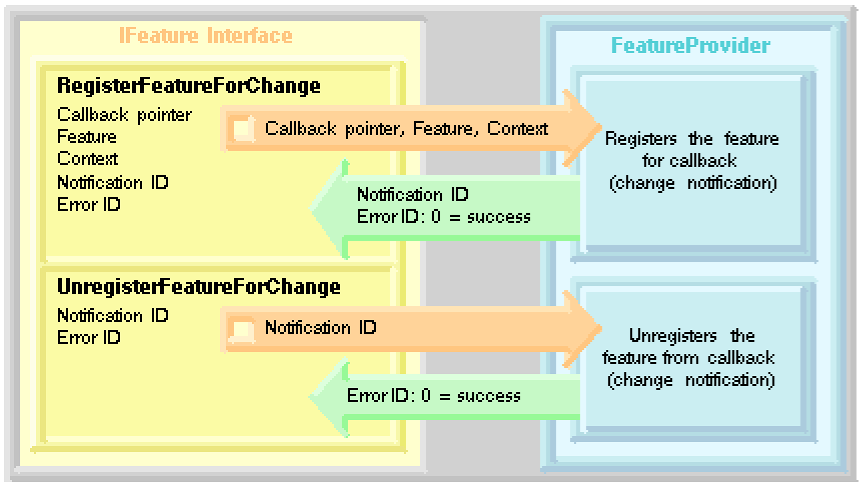
where:

- FeatureName defines the feature to be modified by this operation.
- AttributeNames is an array of names for the attributes associated with the selected feature.
- AttributeValues is an array of values for the attributes of the selected feature..
- pErrorID is a pointer to the error number. If the error number is not 0, the operation was not successful. A value of 0 for pErrorID does not always indicate success. See Programming Tips and Error Handling.

You use the RegisterFeatureForChange method to register a specific feature for change notification (callback). The Feature Provider generates a call to the OnFeatureChanged method whenever a value for any of the Read or Read/Write attributes of the registered feature changes. You program the OnFeatureChanged method with custom software to respond to changes in the features that you registered. Changes to Write attributes do not generate a callback.

You use the UnregisterFeatureForChange method to remove the registration for a selected feature.

Not all features can be registered. Refer to the list of features to determine whether a feature can be registered for callback.


IFeature Interface
RegisterFeatureForChange
Callback pointer
Feature
Context
Notification ID
Error ID
Callback pointer, Feature, Context
Notification ID
Error ID: 0 = success
UnregisterFeatureForChange
Notification ID
Error ID
Notification ID
Error ID: 0 = success
FeatureProvider
Registers the feature for callback (change notification)
Unregisters the feature from callback (change notification)

RegisterFeatureForChange

The RegisterFeatureForChange method registers a specified feature with the Feature Provider to generate notification whenever the values of the attributes of that feature change. You pass the IFeatureCallback interface pointer to the Feature Provider, and the Feature Provider performs the callback on the OnFeatureChanged method that you implemented.

The RegisterFeatureForChange method generates a pointer to a unique notification ID for the registered feature as an output parameter.

RegisterFeatureForChange (pCallback, FeatureName, Context, pNotificationID, pErrorID)

Visual Basic

```
Sub RegisterFeatureForChange(
   pCallback As IFeatureCallback,
   FeatureName As String,
   Context,
   pNotificationID As Long,
   pErrorID As Long)
```

Visual C++

```
HRESULT RegisterFeatureForChange(
   [in] IFeatureCallback* pCallback,
   [in] BSTR FeatureName,
   [in] VARIANT Context,
   [in,out] long* pNotificationID,
   [in,out] long* pErrorID);
```

C#

```
instance void  RegisterFeatureForChange(
   [in] class    S7UnifiedPanel.IFeatureCallback  marshal( interface)
pCallback,
   [in] string marshal( bstr) FeatureName,
   [in] object marshal( struct) Context,
   [in][out] int32& pNotificationID,
   [in][out] int32& pErrorID) runtime managed internalcall
```

Borland Delphi 7

```
procedure RegisterFeatureForChange(
   const pCallback: IFeatureCallback;
   const FeatureName: WideString; Context: OleVariant;
   var pNotificationID: Integer;
   var pErrorID: Integer); safecall;
```

where:

- pCallback is a pointer to a callback interface IFeatureCallback.
- FeatureName is the name of the feature to be registered by this operation.
- Context is a pointer to a client context that is stored and returned by the OnFeatureChanged method. This allows your application to identify elements within a specific feature.
- pNotificationID is a pointer to a unique identification number. You use this identification number when unregistering the feature.
- pErrorID is a pointer to the error number. If the error number is not 0, the operation was not successful.

UnregisterFeatureForChange

The UnregisterFeatureForChange method sends the notification ID to remove the callback registration for the specified feature.

> **Note:** Ensure that your client application unregisters all of the registered features and disconnects from the Feature Provider before exiting.

UnregisterFeatureForChange (NotificationID, pErrorID)

Visual Basic

```
Sub UnregisterFeatureForChange(
   NotificationID As Long,
   pErrorID As Long)
```

Visual C++

```
HRESULT UnregisterFeatureForChange(
   [in] long NotificationID,
   [in,out] long* pErrorID);
```

C#

```
instance void  UnregisterFeatureForChange(
   [in] int32 NotificationID,
   [in][out] int32& pErrorID) runtime managed internalcall
```

Borland Delphi 7

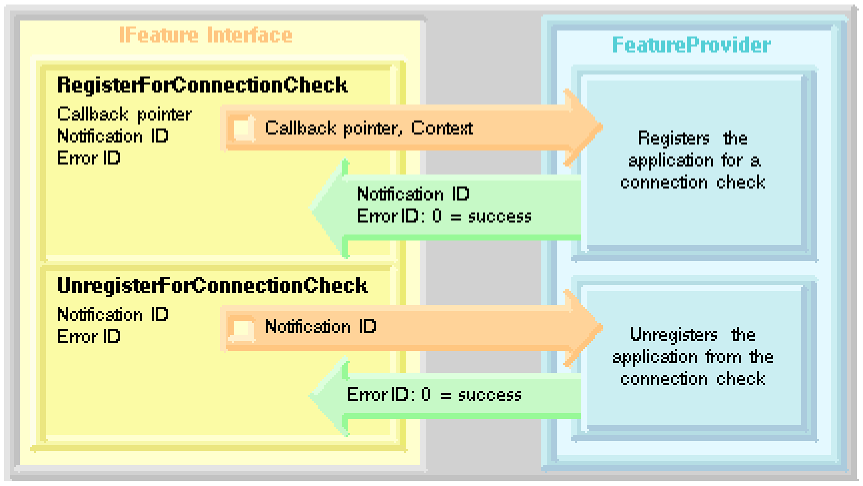
```
procedure UnregisterFeatureForChange(
   NotificationID: Integer;
   var pErrorID: Integer); safecall;
```

where:

- NotificationID is the identification number that was returned by the RegisterFeatrueForChange method. In order to ensure that the same feature is resigistered or unregistered, the NotificationID used by the call to UnregisterFeatureForChange must be the same as the Notification ID that was obtained from the call to RegisterFeatureForChange.
- pErrorID is a pointer to the error number. If the error number equals 0, the operation was successful.

You use the RegisterForConnectionCheck to enable the Feature Provider to perform a connection check. If the connection is broken, the Feature Provider generates a call to the OnPLCDisconnect method of the IFeatureCallback interface. You program the OnPLCDisconnect method with custom software for responding to a loss of PLC connection.

You use the UnregisterForConnectionCheck method to disable the connection check.


IFeature Interface
FeatureProvider
RegisterForConnectionCheck
Callback pointer
Notification ID
Error ID
Callback pointer, Context
Registers the application for a connection check
Notification ID
Error ID: 0 = success
UnregisterForConnectionCheck
Notification ID
Error ID
Notification ID
Unregisters the application from the connection check
Error ID: 0 = success

RegisterForConnectionCheck

The RegisterForConnectionCheck method sets up a callback if the connection to the PLC is broken. The RegisterForConnectionCheck method generates a pointer to a unique notification ID for the registered connection check as an output parameter.

RegisterForConnectionCheck (pCallback, pNotificationID, pErrorID)

Visual Basic

```
Sub RegisterForConnectionCheck(
   pCallback As IFeatureCallback,
   pNotificationID As Long,&
   pErrorID As Long)
```

Visual C++

```
HRESULT RegisterForConnectionCheck(
   [in] IFeatureCallback* pCallback,
   [in,out] long* pNotificationID,
   [in,out] long* pErrorID);
```

C#

```
instance void RegisterForConnectionCheck(
   [in] class S7UnifiedPanel.IFeatureCallback marshal( interface)
pCallback,
   [in][out] int32& pNotificationID,
   [in][out] int32& pErrorID) runtime managed internalcall
```

Borland Delphi 7

```
procedure  RegisterForConnectionCheck(
   const pCallback: IFeatureCallback;
   var pNotificationID:   Integer;
   var pErrorID: Integer); safecall;
```

where:

- pCallback is a pointer to a callback interface IFeatureCallback.
- pNotificationID is a pointer to a unique identification number. You use this identification number when unregistering the connection check.
- pErrorID is a pointer to the error number. If the error number is not 0, the operation was not successful.

UnregisterForConnectionCheck

The UnregisterForConnectionCheck ends the connection check.

UnregisterForConnectionCheck (NotificationID, pErrorID)

Visual Basic

```
Sub UnregisterForConnectionCheck(
   NotificationID As Long,
   pErrorID As Long)
```

Visual C++

```
HRESULT UnregisterForConnectionCheck(
   [in] long NotificationID,
   [in,out] long* pErrorID);
```

C#

```
instance void  UnregisterForConnectionCheck(
   [in] int32 NotificationID,
   [in][out] int32& pErrorID) runtime managed internalcall
```

Borland Delphi 7

```
procedure UnregisterForConnectionCheck(
   NotificationID: Integer;
   var pErrorID: Integer); safecall;
```

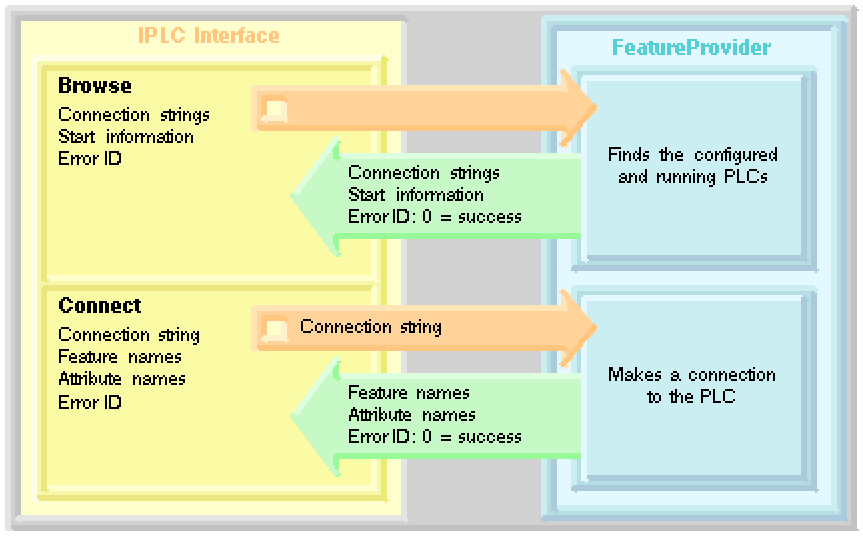
where:

- NotificationID is the identification number that was returned by the RegisterForConnectionCheck method.
- pErrorID is a pointer to the error number. If the error number equals 0, the operation was successful.

Methods of the IPLC Interface

The IPLC interface provides two methods that allow the client application to perform the following tasks:

- Find all PC-based PLC installed on the computer (Browse)
- Connect to a specific PLC (Connect)



Browse

You use the Browse method to find the available PLCs that your client application can access. The Browse method returns arrays of connection strings and start information for any PC-based PLC on the computer.

Browse (pConnectStrings, pStartInfos, pErrorID)

Visual Basic

```
Sub Browse(
   pConnectStrings,
   pStartInfos,
   pErrorID As Long)
```

Visual C++

```
HRESULT Browse(
   [in,out] VARIANT* pConnectStrings,
   [in,out] VARIANT* pStartInfos,
   [in,out] long* pErrorID);
```

C#

```
instance void  Browse(
   [in][out] object&  marshal( struct) pConnectStrings,
   [in][out] object&  marshal( struct) pStartInfos,
   [in][out] int32& pErrorID) runtime managed internalcall
```

Borland Delphi 7

```
procedure Browse(
   var pConnectStrings: OleVariant;
   var pStartInfos: OleVariant;
   var pErrorID: Integer); safecall;
```

where:

- pConnectStrings is a pointer to the connection information for the PLC that provides the name of the computer (ComputerName), the name of the PLC (PLCName), and the type of PLC (PLCType). A slash ("/") or backslash ("\") separates each element of the connection information.
- pStartInfos is a pointer to the date and time that the PLC started.
  - If the PLC has not started, the start information is empty.
  - If the PLC is running, the start information returns the following value: Running
  - If the PLC has been configured but is not running, the start information returns the following value: Configured
- pErrorID is a pointer to the error number. If the error number is not 0, the operation was not successful. A value of 0 for pErrorID does not always indicate success. See Programming Tips and Error Handling.

Connect

The Connect method connects to the specified PLC and returns the list of features that are supported by that PLC.

Connect (ConnectString, pIFeature, pFeatureNames, pAttributeNamesArray, pErrorID)

Visual Basic

```
Sub Connect(
   ConnectString As String,
   pIFeature As IFeature,
   pFeatureNames,
   pAttributeNamesArray,
   pErrorID As Long)
```

Visual C++

```
HRESULT Connect(
   [in] BSTR ConnectString,
   [in,out] IFeature** pIFeature,
   [in,out] VARIANT* pFeatureNames,
   [in,out] VARIANT* pAttributeNamesArray,
   [in,out] long* pErrorID);
```

C#

```
instance void  Connect(
   [in] string  marshal( bstr) ConnectString,
   [in][out] class S7UnifiedPanel.IFeature&  marshal( interface)
pIFeature,
   [in][out] object&  marshal( struct) pFeatureNames,
   [in][out] object&  marshal( struct) pAttributeNamesArray,
   [in][out] int32& pErrorID) runtime managed internalcall
```

Borland Delphi 7

```
procedure Connect(
   const ConnectString: WideString;
   var pIFeature: IFeature;
   var pFeatureNames: OleVariant;
   var pAttributeNamesArray: OleVariant;
   var pErrorID: Integer); safecall;
```

where:

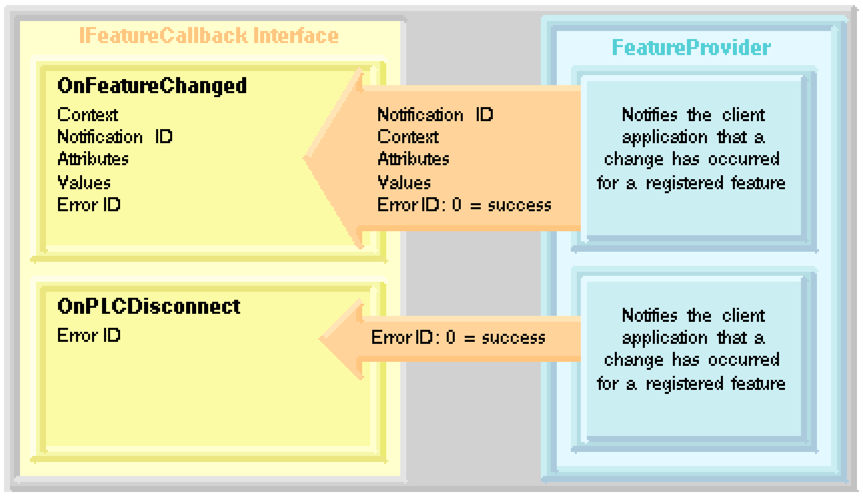
- ConnectString is a string containing the connection information for the PLC that provides the name of the computer (ComputerName), the name of the PLC (PLCName), and the type of PLC (PLCType). A slash ("/") or backslash ("\") separates each element of the connection information.
- pIFeature is a pointer to the IFeature interface
- pFeatureNames is a pointer to an array of features that are supported by the PLC.
- pAttributeNamesArray is a pointer to an array that contains the attribute names associated with a feature.
- pErrorID is a pointer to the error number. If the error number is not 0, the operation was not successful. A value of 0 for pErrorID does not always indicate success. See Programming Tips and Error Handling.

Methods of the IFeatureCallback Interface

The IFeatureCallback interface provides two methods that allow the client application to interact with the PLC. These methods respond to the registration methods (RegisterFeatureForChange and RegisterForConnectionCheck) of the IFeature interface and allow you to perform the following tasks:

- Read the changes to the attributes of a registered feature (OnFeatureChanged)
- Receive notification of connection loss from the Feature Provider (OnPLCDisconnect)

You must write the code in your application to handle these methods.


IFeatureCallback Interface
OnFeatureChanged
Context
Notification ID
Attributes
Values
Error ID
Notification ID
Context
Attributes
Values
Error ID: 0 = success
FeatureProvider
Notifies the client application that a change has occurred for a registered feature
OnPLCDisconnect
Error ID
Error ID: 0 = success
Notifies the client application that a change has occurred for a registered feature

OnFeatureChanged

The Feature Provider calls the OnFeatureChanged method of your client application whenever a Read or Read/Write attribute value for a registered feature changes. This method provides the notification ID of the registered feature and the values for the attributes of that feature. You implement code in the OnFeatureChanged method to respond to the changes. Changes to attributes of type Write do not generate a callback.

OnFeatureChanged (FeatureName, Context, NotificationID, AttributeNames, AttributeValues)

Visual Basic

```
Sub OnFeatureChanged(
   FeatureName As String,
   Context,
   NotificationID As Long,
   AttributeNames,
   AttributeValues)
```

Visual C++

```
HRESULT OnFeatureChanged(
   [in] BSTR FeatureName,
   [in] VARIANT Context,
   [in] long NotificationID,
   [in] VARIANT AttributeNames,
   [in] VARIANT AttributeValues);
```

C#

```
instance void  OnFeatureChanged(
   [in] string  marshal( bstr) FeatureName,
   [in] object  marshal( struct) Context,
   [in] int32   NotificationID,
   [in] object  marshal( struct) AttributeNames,
   [in] object  marshal( struct) AttributeValues) runtime managed
internalcall
```

Borland Delphi 7

```
procedure OnFeatureChanged(
   const FeatureName: WideString; Context: OleVariant;
   NotificationID: Integer;
   AttributeNames: OleVariant;
   AttributeValues: OleVariant); safecall;
```

where:

- FeatureName is the name of the registered feature that changed.
- Context contains the value of the Context that was passed by the RegisterFeatureForChange method.
- NotificationID is the notification identifier that was returned by the RegisterFeatureForChange method.
- AttributeNames contains the names of the attributes associated with the feature.
- AttributeValues contains the values for the attributes of the feature.

OnPLCDisconnect

The Feature Provider calls the OnPLCDisconnect method of the client application when the connection to the PLC has been broken and the client application has registered for notification of connection loss. You use the RegisterForConnectionCheck method to register for PLC connection loss notification. You implement code in the OnPLCDisconnect method to respond to a broken PLC connection.

OnPLCDisconnect (ErrorID)

Visual Basic

```
Sub OnPLCDisconnect(ErrorID As Long)
```

Visual C++

```
HRESULT OnPLCDisconnect([in] long ErrorID);
```

C#

```
instance void OnPLCDisconnect(
      [in] int32 ErrorID) runtime managed internalcall
```

Borland Delphi 7

```
procedure OnPLCDisconnect(ErrorID: Integer); safecall;
```

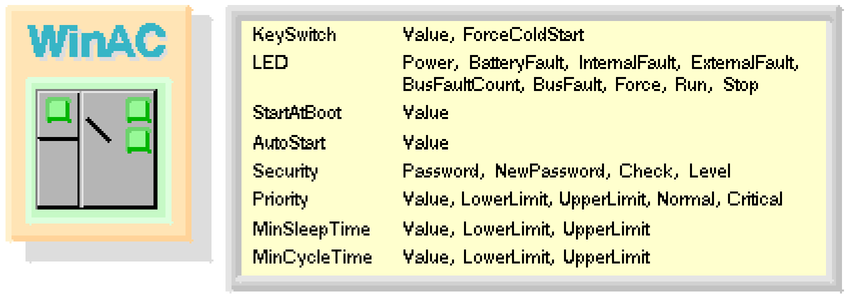
where:

- ErrorID denotes the error condition that resulted in the loss of PLC connection.

Using the Features of the PLC

Each PLC supports a set of features with specific attributes. Some attributes only allow you to read the value, while others allow you to change the value of that attribute. In addition, some attributes support callback, allowing your client application to be notified whenever the value of that attribute changes.

The Feature Provider interacts with the PLC by reading or writing the values of the attributes for the features that are supported by the PLC. Each PLC supports a set of features that represent an element or operation of the PLC, such as a keyswitch or status LEDs. The feature has a set of attributes that contain a value.



KeySwitch	Value, ForceColdStart
LED	Power, BatteryFault, InternalFault, ExternalFault, BusFaultCount, BusFault, Force, Run, Stop
StartAtBoot	Value
AutoStart	Value
Security	Password, NewPassword, Check, Level
Priority	Value, LowerLimit, UpperLimit, Normal, Critical
MinSleepTime	Value, LowerLimit, UpperLimit
MinCycleTime	Value, LowerLimit, UpperLimit

For example, the mode switch that sets the operating mode of a PC-based PLC is controlled by the Value attribute of the KeySwitch feature. The Value attribute can be set to the following values:

- STOP: Sets the mode switch to STOP and places the PLC in STOP mode
- RUN: Sets the mode switch to RUN and places the PLC in RUN mode
- RUNP: Sets the mode switch to RUN-P and places the PLC in RUN mode
- MRES: Initiates a memory reset of the PLC memory

Different PLCs support a different set of features. For example, WinLC Basis and WinLC RTX support a set of features and attributes for tuning the performance of the PLC within the Windows operating system. These features are not supported by a slot PLC because the slot PLC operates independently from the Windows operating system.

The features for the PLCs can be grouped according to functions. For more information, see the List of Features and Attributes.

Features That Correspond to the Operations of the PLC

Feature	**Attribute**	**Description**
KeySwitch	Value, ForceColdStart	Changes the keyswitch settings for changing the operating mode of the PLC Also allows you to force a WinLC controller to perform a cold restart
LED	Power, BatteryFault, InternalFault, ExternalFault, BusFaultCount, BusFault_, Force, Run, Stop	Provides the state of the status indicators for the PLC
FMR	*(no attribute)*	Performs a Failure Message Reset (FMR) for a slot PLC
Error	ID	Provides the error code from the PLC for an illegal operation, such as attempting to restore an archive file when the keyswitch is set to RUN.

Features That Start or Shut down the PLC

Feature	**Attribute**	**Description**
PLCInstance	Value	Creates or shuts down an instance of a WinLC controller Cannot be used with the PLCPower feature
PLCPower	Value	Turns the power on or off for a slot PLC Cannot be used with the PLCInstance feature
PLC	Value	Reports the state of the PLC: • Started or powered on • Shutdown or powered off • Not available
Personality	Name, Type, ProductCode, SWRelease, FWRelease, HWRelease	Provides the product-specific information about the PLC
StartAtBoot	Value	Configures the WinLC controller to automatically start whenever the computer is booted (turned on or restarted)
MemoryCardFile	Buffer, Size	Contains the PLC archive/restore file (memory card file

Features That Configure Options for the PLC

Feature	Attribute	Description
AutoStart	Value	Sets the Autostart option for starting a WinLC controller
AutoLoad	Value, KeySwitch, TargetFile	Sets the Autoload option for starting a slot PLC
CPULanguage	CurrentLanguage, Count, Language_	Provides the language selection for the PLC
Security	n/a	Internal use only: This feature is not available for CMI applications.

Features That Tune the Performance of a WinLC Controller

The following features are used for WinLC controllers only and are not applicable for a slot PLC.

Feature	Attribute	Description
Priority	Value, LowerLimit, UpperLimit, Normal, Critical	Sets the parameters for adjusting the priority of a WinLC controller
MinCycleTime	Value, LowerLimit, UpperLimit	Sets the parameters for the minimum cycle time of a WinLC controller
MinSleepTime	Value, LowerLimit, UpperLimit	Sets the parameters for the minimum sleep time of a WinLC controller
OBExecution	WakeInterval, SleepInterval, DefaultWakeInterval, DefaultSleepInterval, UpperLimit, LowerLimit	Provides information about the amount of sleep time for a WinLC controller
Timing	UpperLimit, CycleTimeCount, CycleTimeBuffer, CycleTimeMax. CycleTimeAverage, CycleTimeLast, ExecTimeMin, ExecTimeMax, ExecTimeAverage, ExecTimeLast, SleepIntervalCounter, Clear	Provides information about the performance for a WinLC controller (showing the configured and actual execution times of the control program)
Usage	PC, PLC, CPUCount, CPU_	Provides information about the percentage of CPU usage for the computer running a WinLC Basis controller Not applicable for WinLC RTX

Features That Relate to the Diagnostic Buffer of the PLC

Feature	Attribute	Description
Diagnostic	Language, Count, Time_, Date_, EventShort_, EventLong_, EventID_, EventHex_	Provides the information in the diagnostic buffer for the PLC
DiagnosticLanguage	Count, Language_	Provides the language options for the diagnostic buffer

List of Features and Attributes

The Feature Provider interacts with the PLC by reading or writing the values of the attributes for the features described in this topic. Some attributes are read-only, some attributes are write-only, and other attributes allow your client application to both read and write values. In addition, some features support callback, which allows your client application to be notified whenever the value of any attribute in that feature changes. The callback capability allows your client application to respond to PLC events.

Different PLCs support different sets of features. The Connect method of the IFeature interface returns a list of the features and attributes supported by a specific PLC.

CMI provides a header file for each of the programming environments that contains the feature names, attribute names, and constant values for specified attributes. Use the constants as defined in the header file for your programming environment when constructing your CMI function calls. The header files for the features are listed below:

• Visual Basic 6.0:	Feature.bas
• Visual Basic .NET:	Feature.vb
• Visual C++ (6.0 and .NET):	featureStrDefine.h
• Visual C# .NET:	Features.cs
• Borland Delphi 7:	S7UP_Feature.pas

About Attribute Types

Most feature attributes are of type Read, Write, or Read/Write. They represent data that is read from or written to the PLC. Some attributes, however, can be typed as Input or Output parameters.

An attribute of type Input can be used as an input parameter to a GetFeature or SetFeature call and provides information about executing the call. An attribute of type Output is returned as an output parameter from either a GetFeature or SetFeature call, for example a return status. Input and Output parameters are not read from or written to the PLC.

The following table describes each of the attribute types:

Attribute Type	Description
Read	Value to be read from the PLC using a GetFeature call, for example, Error feature, ID attribute
Write	Value to be written to the PLC using a SetFeature call, for example, PLCInstance or PLCPower feature, Value attribute
Read/Write	Value that can be either read from the PLC using a GetFeature call, or written to the PLC using a SetFeature call, for example, Autostart feature, Value attribute
Input	Parameter required by either a GetFeature or SetFeature call to provide information about how to perform the call, for example, Diagnostic feature, Language attribute The Language attribute when passed to a GetFeature call for the Diagnostic feature, specifies the language in which GetFeature is to return the diagnostic information. If Language is VAL_LANGUAGE_GERMAN, for example, the GetFeature returns the event descriptions in German. The input attribute Language is not read from or written to the PLC or to the diagnostic buffer language setting. It is only input information for the GetFeature call.
Output	Parameter returned from either a GetFeature or SetFeature call that provides information about the call, for example, a status value. Currently, CMI does not define any attributes of type Output.

About Attribute Values

The attribute values in the list of features and attributes are either string or integer data types. Use the data types of your programming environment for declaring variables for attribute values.

In the list of features and attributes, the value column sometimes lists a fixed set of values that are valid for the specific attribute, especially when the attribute value is a string. In this case, the value column provides the string constants from the feature header file. The names of the string constants and their values are the same for each programming environment; for example, VAL_ON is "On" in the feature header file for each supported programming environment.

In other cases, the value column provides a range of valid values for the attribute, for example, 0..100 for a percentage.

In still other cases, the value column describes the requirements or limitations for the value for the specific attribute, for example, the relationship between the minimum sleep time for a PLC and the scan cycle monitoring time configured in STEP 7.

When the value column is empty, the attribute value has no restrictions other than its data type.

About Repeating Attributes

Several of the features contain attributes for which GetFeature returns multiple occurrences. In the attribute name array returned by the Connect method, these attribute names end in "_" and occur only once.

When GetFeature returns a feature with multiply occurring attributes, it appends an integer 0..n to the repeating attribute names to number the multiple occurrences. For example, a call to GetFeature for the CPULanguage feature returns Language_0, Language_1, and Language_2 for the three languages supported by the connected CPU (English, German, and French.)

The complete set of features and attributes of the Controller Management Interface is described below:

AutoStart

Callback supported

Valid for PLC type: WinAC Basis, WinAC RTX

This feature contains the Autostart option that defines the operating mode for the PLC when it starts. If Autostart is set to On, the controller starts up in the operating mode that it was in when it was shut down. If Autostart is set to Off, the controller starts up in STOP mode. The values listed below are string constants defined in the feature header file for your environment.

Attribute	Data Type	Type	Value	Description
Value	String	Read/Write	VAL_ON VAL_OFF	Specifies whether the AutoStart option is configured or not

AutoLoad

Valid for PLC type: WinAC Slot

This feature contains the Autoload option for starting the slot PLC. This allows you to specify the keyswitch position and to load a specific control program when power is turned on for the slot PLC. The control program must be in a file named by the TargetFile attribute.

Attribute	**Data Type**	**Type**	**Value**	**Description**
Value	String	Read/Write	VAL_ON VAL_OFF	Specifies whether the AutoLoad option is configured or not
KeySwitch	String	Read/Write	VAL_AUTOLOAD_KS_STOP VAL_AUTOLOAD_KS_RUN VAL_AUTOLOAD_KS_RUNP	Keyswitch position (Operating Mode selection)
Buffer	n/a	n/a		Internal use only: This attribute is not available for CMI applications.
BufferSize	n/a	n/a		Internal use only: This attribute is not available for CMI applications.
TargetFile	String	Read/Write		Name of file that contains the control program to download on power up

ControllerHelp

Valid for PLC type: WinAC Basis, WinAC RTX, WinAC Slot

This read-only feature provides information about the online documentation for the PLC.

The WinAC controllers provide online documentation in the following electronic formats: WinHelp (Windows help system, based on an RTF file format), HTMLHelp (an HTML-based help system that is compiled into a CHM file), and WebHelp (a generic HTML-based help system).

The WinAC controllers support the following languages: German, English, and French.

Attribute	**Data Type**	**Type**	**Value**	**Description**
Host	String	Read		Computer name where PLC and help system are installed
HelpSystem	String	Read	VAL_HELPSYSTEM_WIN VAL_HELPSYSTEM_WEB VAL_HELPSYSTEM_HTML	Type of help system: WinHelp, HTMLHelp, or WebHelp
HelpDir	String	Read		Full pathname to PLC help system
Count	Integer	Read		Number of available languages the PLC supports for displaying help system
Language_	String	Read	VAL_LANGUAGE_ENGLISH VAL_LANGUAGE_GERMAN VAL_LANGUAGE_FRENCH	Available languages the PLC supports for displaying help system No order is implied.

CPULanguage

Valid for PLC type: WinAC Basis, WinAC RTX

This feature contains the current language setting for the PLC, as well as the number of supported languages, and an array of those languages.

The WinAC controllers support the following languages: German, English, and French.

Attribute	Data Type	Type	Value	Description
CurrentLanguage	String	Read/Write	VAL_LANGUAGE_ENGLISH VAL_LANGUAGE_GERMAN VAL_LANGUAGE_FRENCH	Language setting for the PLC
Count	Integer	Read		Number of available languages
Language_	String	Read	VAL_LANGUAGE_ENGLISH VAL_LANGUAGE_GERMAN VAL_LANGUAGE_FRENCH	Available languages No order is implied.

Diagnostic

Valid for PLC Type: WinAC Basis, WinAC RTX, WinAC Slot

This read-only feature accesses the information in the diagnostic buffer of the PLC.

The WinAC controllers support the following languages: German, English, and French. See also the DiagnosticLanguage feature.

Attribute	**Data Type**	**Type**	**Value**	**Description**
Language	String	Input	VAL_LANGUAGE_ENGLISH VAL_LANGUAGE_GERMAN VAL_LANGUAGE_FRENCH	Input parameter to GetFeature that defines the language in which to return the diagnostic buffer
Count	Integer	Read		Number of diagnostic buffer event entries
Time_	String	Read		Time of event, for each event
Date_	String	Read		Date of event, for each event
EventShort_	String	Read		Short description of event, for each event
EventLong_	String	Read		Long description of event, for each event
EventID_	String	Read		ID of event, for each event
EventHex_	String	Read		Hexadecimal ID of event, for each event

DiagnosticLanguage

Valid for PLC Type: WinAC Basis, WinAC RTX, WinAC Slot

This read-only feature provides information about the language options for the diagnostic buffer.

Attribute	Data Type	Type	Value	Description
Count	Integer	Read		Number of available languages for diagnostic buffer
Language_	String	Read	VAL_LANGUAGE_ENGLISH VAL_LANGUAGE_GERMAN VAL_LANGUAGE_FRENCH VAL_LANGUAGE_SPANISH VAL_LANGUAGE_ITALIAN VAL_LANGUAGE_JAPANESE	Available languages for diagnostic buffer No order is implied.

Error

Callback supported

Valid for PLC Type: WinAC Basis, WinAC RTX, WinAC Slot

This read-only feature returns the error code from the PLC for an illegal operation, such as attempting to restore an archive file when the keyswitch is set to RUN.

Attribute	Data Type	Type	Description
ID	Integer	Read	Error code returned from PLC

The following table lists the possible error ID values and descriptions.

Error ID and Description
_PSERR_OKAY (= 0) The operation was executed successfully.
_PSERR_NO_MEMORY (= 1) The PLC does not have sufficient memory to perform this operation.
_PSERR_ARCHIVE_NOT_VALID_IN_RUN (= 2) The PLC must be in STOP mode before you can create an archive file of the control program.
_PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU (=3) The PLC was unable to create the archive file.
_PSERR_RESTORE_CANNOT_LINKIN_BLOCK (= 4) The PLC is unable to restore the control program from the archive file.

Error ID and Description
_PSERR_RESTORE_NOT_VALID_IN_RUN (= 5) The PLC must be in STOP mode before you can restore an archived control program.
_PSERR_RESTORE_FILE_INVALID (= 6) The file specified is not an archive file or has become corrupted.
_PSERR_INIT_EDBSERVER (= 7) The diagnostic buffer was not able to access the descriptions and event data list from the EDB server. You may need to reinstall the PLC.
_PSERR_INIT_PDH (= 8) The system information of the computer (such as CPU usage and number of processors) in not available. This information is provided by the PDH.DLL of the Windows operating system.
_PSERR_KEYSWITCH_NOT_ALLOWED_IN_ MCF_OP (= 9) There was an attempt to change the setting of the keyswitch during an archive or restore operation.
_PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU_PASSWORD_PROTECTED (= 10) The security setting for the PLC requires a password before allowing the control program to be archived.
_PSERR_FILE_SIZE_EXCEEDED File size exceeds maximum allowed size.
_PSERR_FWUPDATE_NOT_SUCCEEDED An attempt to update the firmware did not succeed.
_PSERR_FWUPDATE_NOT_POSSIBLE An attempt to update the firmware is not possible, possibly due to a corrupted buffer.

Firmware Update

Valid for PLC Type: WinAC Slot

Internal use only: This feature is not available for CMI applications.

FMR

Valid for PLC type: WinAC Slot

This write-only feature performs the Failure Message Reset (FMR) for the slot PLC. It has no attributes.

To perform a Failure Message Reset, the CMI application must call SetFeature with the FeatureName defined by the constant FEATURE_FMR. The CMI application must declare variables of the correct data types for AttributeNames and AttributeValues and pass these variables to the SetFeature call. The FMR feature has no attributes, but the SetFeature call does require the parameters AttributeNames and AttributeValues. The CMI application does not have to assign any values to these parameters; it only has to pass parameters of the correct data types expected by the SetFeature method.

KeySwitch

Callback supported

Valid for PLC type: WinAC Basis, WinAC RTX, WinAC Slot

This feature changes the keyswitch settings for changing the operating mode of the PLC:

- MRES initiates a memory reset.
- STOP places the PLC in STOP mode. With the keyswitch set to STOP, the PLC does not execute the control program, and the outputs are set to their safe states. STEP 7 can modify the control program and reset the memory (MRES), but cannot change the operating mode of the PLC.
- RUN places the PLC in RUN mode. With the keyswitch set to RUN, the PLC executes the control program. STEP 7 can monitor the control program, but cannot change the variables, reset the memory (MRES), or change the operating mode of the PLC.
- RUNP places the PLC in RUN mode and allows STEP 7 to interact with the PLC. With the keyswitch set to RUN-P, STEP 7 can monitor the control program, change the variables, reset the memory (MRES), and change the operating mode of the PLC.

The optional ForceColdstart attribute allows you to force a WinLC controller to perform a cold restart instead of a warm restart. A restart deletes the peripheral I/O (PII and PIQ), deletes the non-retentive memory bits (M), timers (T) and counters (C), and changes the peripheral outputs to a pre-defined safe state (default is 0).

- A warm restart saves the retentive memory bits, timers, counters and data blocks (DBs).
- A cold restart does **not** save the retentive memory bits, timers, counters and data blocks (DBs), but sets these areas to their default (initial) values.

Attribute	**Data Type**	**Type**	**Value**	**Description**
Value	String	Read/Write	VAL_KEYSWITCH_MRES VAL_KEYSWITCH_STOP VAL_KEYSWITCH_RUN VAL_KEYSWITCH_RUNP	Specifies keyswitch position (operating mode setting) of PLC
ForceColdstart (optional)	String	Write	VAL_ON VAL_OFF	Specifies whether or not to perform cold restart when PLC is restarted

LED

Callback supported

Valid for PLC Type: WinAC Basis, WinAC RTX, WinAC Slot

This read-only feature accesses the status indicators (or LEDs) for the PLC. Each LED supports the following states: ON (on, not blinking), OFF (off, not blinking), Blinking2HZ (blinking on and off slowly at a speed of 2 Hz), and Blinking05HZ (blinking on and off quickly, at a speed of 0.5 Hz).

Note: The Run and Stop LEDs display the actual operating mode of the PLC. The value of the KeySwitch feature (RUN, RUNP, or STOP) determine the position of the keyswitch or mode button, which can differ from the actual operating mode of the PLC.

To reset the status indicators related to the power supply of the PLC, see the FMR feature.

Attribute	Data Type	Type	Value	Description
Power	String	Read	VAL_LED_ON VAL_LED_OFF VAL_LED_BLINKING2HZ VAL_LED_BLINKING05HZ	Indicates that power is turned on for the PLC
BatteryFault (WinAC Slot only)	String	Read	VAL_LED_ON VAL_LED_OFF VAL_LED_BLINKING2HZ VAL_LED_BLINKING05HZ	Indicates a battery fault in the power supply for the PLC
InternalFault	String	Read	VAL_LED_ON VAL_LED_OFF VAL_LED_BLINKING2HZ VAL_LED_BLINKING05HZ	Indicates an error condition that exists within the PLC Examples: a programming error, an arithmetic error, a timer error, or a counter error.
ExternalFault	String	Read	VAL_LED_ON VAL_LED_OFF VAL_LED_BLINKING2HZ VAL_LED_BLINKING05HZ	Indicates an error condition that exists outside of the PLC Examples: a hardware fault, a parameter assignment error, a communication error, or an I/O fault error
BusFaultCount	Integer	Read	VAL_LED_ON VAL_LED_OFF VAL_LED_BLINKING2HZ VAL_LED_BLINKING05HZ	Provides the number of communication channels for calculating the number of Bus Fault LEDs
BusFault_ (optional)	String	Read	VAL_LED_ON VAL_LED_OFF VAL_LED_BLINKING2HZ VAL_LED_BLINKING05HZ	Indicates a fault condition in the communication with the distributed I/O
Force (WinAC Slot only)	String	Read	VAL_LED_ON VAL_LED_OFF VAL_LED_BLINKING2HZ VAL_LED_BLINKING05HZ	Indicates that an input or an output has been forced to a some value
Run	String	Read	VAL_LED_ON VAL_LED_OFF VAL_LED_BLINKING2HZ VAL_LED_BLINKING05HZ	Indicates that the PLC is in RUN mode
Stop	String	Read	VAL_LED_ON VAL_LED_OFF VAL_LED_BLINKING2HZ VAL_LED_BLINKING05HZ	Indicates that the PLC is in STOP mode

MemoryCardFile

Valid for PLC Type: WinAC Basis, WinAC RTX, WinAC Slot

This feature enables archiving and restoring of the memory card file for the PLC control program.

To restore a memory card file (archive file), the CMI application must read a memory card file from the hard disk or other media and convert it to a buffer in the format required by the MemoryCardFile feature. To do this, the CMI application must use the function hideSpecialChars() to convert the memory card file contents to a string to be used for the Buffer attribute. The CMI application must then call SetFeature for the MemoryCardFile feature with this converted buffer as the Buffer attribute, and the size of the converted buffer as the Size attribute.

To archive the PLC control program to a memory card file, the CMI application must call GetFeature to read the Buffer and the Size attributes from the PLC. The application must then call unhideSpecialChars() to convert the string format of the buffer into the format required for the memory card file. The CMI application can then write the contents of the buffer returned by unhideSpecialChars() to a memory card file on the hard disk or other media.

Notice
The use of the MemoryCardFile feature requires a significant amount of memory. The entire contents of a .wld file must be managed as a string buffer to perform archive and restore operations. CMI applications cannot archive and restore directly to or from the hard disk or other media. You must ensure that your system has enough available memory for the memory card file string buffer.

The WinAC ODK installation includes the functions hideSpecialChars() and unhideSpecialChars() as well as sample programs that use the MemoryCardFile feature to perform archive and restore operations.

The conversion functions and sample programs for Visual Basic 6.0, Visual Basic .NET, Visual C++ 6.0, Visual C++ .NET, Visual C# .NET, and Borland Delphi 7 are in the ...\Program Files\Siemens\WinAC\ODK\Examples\CMI folder, under the subfolders CMI_ArchiveRestore_VB, CMI_NET_ArchiveRestore_VB, CMI_ArchiveRestore_Cpp, CMI_ArchiveRestore_Cs, and CMI_ArchiveRestore_Delphi.

Attribute	**Data Type**	**Type**	**Value**	**Description**
Buffer	String	Read/Write	Binary information as string	Contents of memory card file (archive file)
Size	Integer	Read/Write		size of memory card file in bytes

MinCycleTime

Callback supported

Valid for PLC Type: WinAC Basis, WinAC RTX

This feature contains values for the minimum cycle time (in milliseconds) for the PLC scan cycle.

Scan cycle time is the number of milliseconds from the start of one cycle to the start of the next cycle, and the execution time is the actual amount of time used by the PLC to update the I/O and to execute the control program. The cycle time value must be greater than the execution time of the scan to provide execution time for any application that has a lower priority than the PLC.

Attribute	Data Type	Type	Value	Description
Value	Integer	Read/Write	Must be less than Scan Cycle Monitoring Time specified in STEP 7 for the PLC properties.	Minimum scan cycle time
LowerLimit	Integer	Read	0	Lowest allowed cycle time
UpperLimit	Integer	Read	Equal to the Scan Cycle Monitoring Time specified in STEP 7 for the PLC properties.	Highest allowed cycle time

MinSleepTime

Callback supported

Valid for PLC Type: WinAC Basis, WinAC RTX

This feature contains the values for the minimum time (in milliseconds) that the PLC sleeps during the execution of OB1 to allow other Windows applications to be executed.

The sleep time determines how much time is available during the free cycle (execution cycle for OB1) to allow higher priority OBs and other applications to use the resources of the computer.

Attribute	Data Type	Type	Value	Description
Value	Integer	Read/Write	Must be less than Scan Cycle Monitoring Time specified in STEP 7 for the PLC properties.	Minimum sleep time
LowerLimit	Integer	Read	0	Lowest allowed sleep time
UpperLimit	Integer	Read	Equal to the Scan Cycle Monitoring Time specified in STEP 7 for the PLC properties.	Highest allowed sleep time

OBExecution

Valid for PLC Type: WinAC Basis, WinAC RTX

This read-only feature provides information about the amount of sleep time for the PLC in microseconds. See the documentation for your controller before changing any of these attribute values.

Attribute	**Data Type**	**Type**	**Value**	**Description**
WakeInterval	Integer	Read/Write		Execution time limit
SleepInterval	Integer	Read/Write		Forced execution sleep time
DefaultWakeInterval	Integer	Read		Default execution time limit
DefaultSleepInterval	Integer	Read		Default forced execution sleep time
UpperLimit	Integer	Read	0..100%	Maximum execution load as a percentage of the scan cycle
LowerLimit	Integer	Read	0..100%	Minimum execution load as a percentage of the scan cycle

Personality

Valid for PLC Type: WinAC Basis, WinAC RTX, WinAC Slot

This read-only feature provides the product-specific information about the PLC.

Attribute	**Data Type**	**Type**	**Description**
Name	String	Read	PLC name (for example, "WinLC")
Type	String	Read	PLC type (for example, "Basis")
ProductCode (order number)	String	Read	Product code (order number) of PLC
SWRelease	String	Read	Software version level of PLC
FWRelease	String	Read	Firmware version level of PLC
HWRelease	String	Read	Hardware version level of PLC
Slot	n/a	n/a	Internal use only: This attribute is not available for CMI applications.
Rack	n/a	n/a	Internal use only: This attribute is not available for CMI applications.
Owner	n/a	n/a	Internal use only: This attribute is not available for CMI applications.
Company	n/a	n/a	Internal use only: This attribute is not available for CMI applications.

PC_PG_Interface

Internal use only: This feature is not available for CMI applications.

PLC

Callback supported

Valid for PLC Type: WinAC Basis, WinAC RTX, WinAC Slot

This read-only feature provides the state of the PLC:

- Created: The PLC has been started or powered on.
- Shutdowned: The PLC has been shut down or powered off.
- NotAvailable: The PLC is no longer available. For example, you can use STEP 7 to change the name of a running instance of the PLC. In this case, the client application can evaluate this feature and then disconnect in order to connect to the new (renamed) instance of the PLC. This feature can also be used to respond to a situation where the PLC has encountered a fatal error and is no longer responsive.

Attribute	Data Type	Type	Value	Description
Value	String	Read	VAL_PLC_CREATED VAL_PLC_SHUTDOWNED VAL_PLC_NOTAVAILABLE	State of PLC

PLCInstance

Valid for PLC Type: WinAC Basis, WinAC RTX

This write-only feature creates or shuts down an instance of a WinLC controller.

Note: You cannot use the PLCInstance feature with the PLCPower feature.

Attribute	Data Type	Type	Value	Description
Value	String	Write	VAL_PLCINSTANCE_CREATE VAL_PLCINSTANCE_SHUTDOWN	Command to create or shutdown a WinLC controller

PLCPower

Valid for PLC Type: WinAC Slot

This write-only feature turns the power on or off for a slot PLC.

Note: You cannot use the PLCPower feature with the PLCInstance feature.

Attribute	Data Type	Type	Value	Description
Value	String	Write	VAL_ON VAL_OFF	Command to power a WinAC Slot PLC on or off.

Priority

Callback supported

Valid for PLC Type: WinAC Basis, WinAC RTX

This feature sets the parameters for adjusting the priority for the execution of the PLC relative to other applications running on your computer.

Setting the priority higher means that the operating system responds to the PLC before executing lower-priority tasks. This results in less jitter in the start times and execution time of the OBs in the control program.

Attribute	Data Type	Type	Value	Description
Value	Integer	Read/Write	WinLC Basis: 1 to 13, 17 to 29 WinLC RTX: 1 to 62	Priority setting for PLC
LowerLimit	Integer	Read	WinLC Basis and WinLC RTX: 1	Lowest possible priority setting
UpperLimit	Integer	Read	WinLC Basis: 29 WinLC RTX: 62	Highest possible priority setting
Normal	Integer	Read	WinLC Basis: 8 WinLC RTX: 50	Normal priority setting
Critical	Integer	Read	WinLC Basis: 16 WinLC RTX: 62	Critical priority setting

Security

Internal use only: This feature is not available for CMI applications.

StartAtBoot

Callback supported

Valid for PLC Type: WinAC Basis, WinAC RTX

This feature configures the PLC to start automatically whenever the computer is rebooted (turned on or restarted).

Attribute	**Data Type**	**Type**	**Value**	**Description**
Value	String	Read/Write	VAL_ON VAL_OFF	Specifies whether or not to start the PLC automatically after a reboot

Timing

Valid for PLC Type: WinAC Basis, WinAC RTX

This feature provides information about the performance for the PLC (showing the configured and actual execution times of the control program).

The execution time is the actual time the controller takes to complete one pass through the instructions of the user program. This includes executing OB1 and updating the I/O. The scan cycle time is the time required to execute the complete scan cycle, which includes the execution of OB1 and the minimum sleep time.

The cycle time buffer is a space-separated list of values where the first value is the cycle time, and the second value is the number of scan cycles that executed at that cycle time. This buffer can be used as a histogram of scan cycle performance.

Use the Clear attribute to reset all of the values in the PLC.

Attribute	**Data Type**	**Type**	**Description**
UpperLimit	Integer	Read	Configured execution time limit, default is 9000 µs
CycleTimeCount	Integer	Read	Number of data pairs in the CycleTimeBuffer
CycleTimeBuffer	Integer Integer ...	Read	Buffer containing cycle time data as described above
CycleTimeMin	Integer	Read	Minimum scan cycle time
CycleTimeMax	Integer	Read	Maximum scan cycle time
CycleTimeAverage	Integer	Read	Average scan cycle time
CycleTimeLast	Integer	Read	Last scan cycle time
ExecTimeMin	Integer	Read	Minimum execution time
ExecTimeMax	Integer	Read	Maximum execution time
ExecTimeAverage	Integer	Read	Average execution time
ExecTimeLast	Integer	Read	Last execution time
SleepIntervalCounter	Integer	Read	Number of times PLC has had a forced execution sleep
Clear	<empty>	Write	Resets all of the collected timing values

Usage

Valid for PLC Type: WinAC Basis

This read-only feature provides information about the percentage of CPU usage for the computer running a WinAC Basis controller. This feature supports multi-processor computers.

Note
The information about CPU usage is not applicable for a WinAC RTX controller.

Attribute	**Data Type**	**Type**	**Value**	**Description**
PC	Integer	Read	0..100%	Percent of total CPU time spent on PC tasks other than the PLC
PLC	Integer	Read	0..100%	Percent of total CPU time used by the PLC
CPUCount	Integer	Read	0..n	Number of processors
CPU_ (optional)	Integer	Read	0..100%	Execution load as a percentage for each processor (CPU) in your system

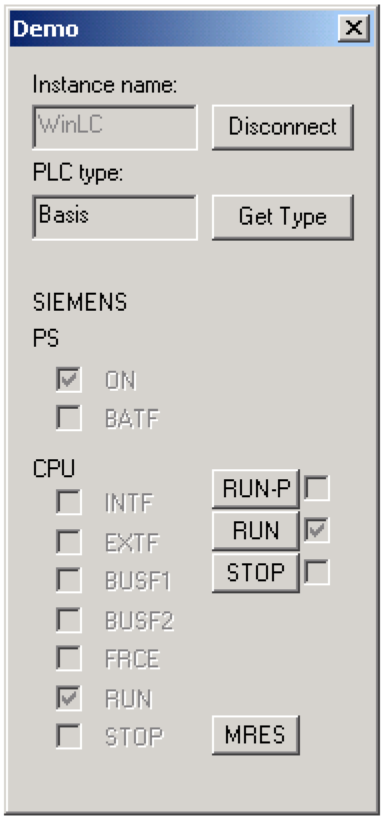
Getting Started

The WinAC Controller Management Interface includes Demo_Panel sample programs that demonstrate how to use the Controller Management Interface to monitor and modify the operation of a WinAC controller.

Each sample program is in a different programming language or enviroment and shows how to use the Controller Management Interface from Visual Basic 6.0, Visual Basic .NET, Visual C++ 6.0, Visual C++ .NET, Visual C# .NET, or Borland Delphi 7. The tasks perfomed in the sample programs are the same types of tasks that you can program in your CMI custom application:

- Include the WinAC Controller Management Interface
- Connect the application to a PLC
- Browse to find available PLCs for connection
- Read and write attributes of PLC features
- Register and respond to changes in a feature
- Disconnect the application from the PLC

Each of the Demo_Panel sample programs uses the Controller Management Interface to implement a control panel application as shown below:



The control panel allows the user to perform these tasks:

- Connect to or disconnect from a PLC that is installed on your computer
- Display the PLC type of the connected controller
- View the status LEDs
- Change the operating mode
- Reset the memory

The sample programs have the same program design and functionality, with the implementation dependent on the programming environment. In brief, each sample program contains the following software:

- Event handlers for each of the buttons on the demo panel.
- Registration for change notification of features that correspond to the LED indicators, the operating modes, the PLC connection status, and the PLC error value.
- OnFeatureChanged implementation to update the control panel display whenever an LED status changes or when the operating mode changes.

WinAC ODK installs the sample programs in the ...\Program Files\Siemens\WinAC\ODK\Examples\CMI directory. When referred to collectively, this manual uses "Demo_Panel" as the sample program name.

The supported development environments and directories containing sample programs for those environments are listed below. Program listings are available in the Reference section.

Programming Development Environment	**Sample Project Folder**
Visual Basic 6.0	CMI_Demo_Panel_VB
Visual Basic .NET	CMI_NET_Demo_Panel_VB
Visual C++ 6.0	CMI_Demo_Panel_Cpp
Visual C++ .NET	CMI_Demo_Panel_Cpp
Visual C# .NET (C sharp)	CMI_Demo_Panel_Cs
Borland Delphi 7	CMI_Demo_Panel_Delphi

Note
The source files for the Visual C++ 6.0 and Visual C++ .NET sample programs are the same and are located in the same folder. This folder has a project workspace for Visual C++ 6.0 and a solution workspace for Visual C++ .NET, both of which include the same set of source files.

Including the Controller Management Interface in a Project

You must include and initialize WinAC Controller Management Interface objects in your development project in order to use the CMI features, attributes, and functions to access a WinAC controller. How to include specific files and set up external references depends upon your programming environment; however, the requirements for including the Controller Management Interface are the same. These requirements include:

- References to the type libraries (.tlb files) for the IPLC and the IFeature Interfaces
- Inclusion of the IFeatureCallback interface
- Definitions of the features and attributes
- Instantiation of the Feature Provider

Referencing the Controller Management Interface Type Libraries

The CMI type libraries provide an interface between your application and the controller through the Feature Provider COM object. You must include these type libraries in your project to establish the IPLC interface and the IFeature interface between your application and a WinAC controller.

To include the CMI type libraries, follow the instructions for your development environment.

Note
The "FeatureProvider 1.0 Type Library" is S7WCUFPX.DLL and the "S7 WinAC Unified Panel Interfaces 1.0 Type Library" is S7WCUIFX.DLL. These DLLs implement the WinAC Controller Management Interface.

Visual Basic 6.0

For Visual Basic 6.0, follow these steps:

1. Select **Project > References** from the project menu.
2. Select the checkboxes for "FeatureProvider 1.0 Type Library" and "S7 WinAC Unified Panel Interfaces 1.0 Type Library".
3. Click OK.

Visual Basic .NET

For Visual Basic .NET, follow these steps:

1. Select **Project > Add Reference** from the project menu.
2. From the COM tab of the Add Reference dialog, select the checkboxes for "FeatureProvider 1.0 Type Library" and "S7 WinAC Unified Panel Interfaces 1.0 Type Library".
3. Click OK.

Visual C++ 6.0 and Visual C++ .NET

For Visual C++, follow these steps:

1. Insert the following two lines in one of your header files, for example, StdAfx.h:

   ```
   #import  "FeatureProvider.tlb" no_namespace
   #import  "Interfaces.tlb" no_namespace exclude("IFeatureCallback")
   ```

   By default, these files are in the ...\Program Files\Siemens\WinAC\ODK\Tlb directory.
2. Provide a path to FeatureProvider.tlb and Interfaces.tlb from your project, following these steps based on your development environment:

   Visual C++ 6.0:

   a. Select the **Project > Settings** menu command.
   b. Select the C/C++ tab and category Preprocessor.
   c. Enter the relative path to FeatureProvider.tlb and Interfaces.tlb in the "Additional include directories" field.

   Visual C++ .NET:

   a. Select the **Project > Properties** menu command.
   b. Select the C/C++ tab and category General.
   c. Enter the relative path to FeatureProvider.tlb and Interfaces.tlb in the "Additional include directories" field.

Visual C# .NET

For Visual C# .NET, follow these steps:

1. Select **Project > Add Reference** from the project menu.
2. From the COM tab of the Add Reference dialog, select the checkboxes for "FeatureProvider 1.0 Type Library" and "S7 WinAC Unified Panel Interfaces 1.0 Type Library".
3. Click OK.

Borland Delphi 7

For Borland Delphi 7, follow these steps:

1. Insert the following code in your main program file to include the S7UP_Interfaces.pas and S7UP_FeatureProvider.pas type libraries:

```
interface

uses  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
ComObj, S7UP_Interfaces, S7UP_FeatureProvider, S7UP_Feature, StdCtrls;
```

2. Provide a path to S7UP_Interfaces.pas and S7UP_FeatureProvider.pas, following these steps:
   a. Select the **Project > Options** menu command.
   b. Select the Directories/Conditionals tab.
   c. Enter the relative path to S7UP_Interfaces.pas and S7UP_FeatureProvider.pas in the "Search Path" field. By default, these files are in the ...\Program Files\Siemens\WinAC\ODK\Examples\Support_Delphi directory.

Including the IFeatureCallback Interface

The IFeatureCallback interface allows an application to define specific features that send a notification event back to the application when they change. When one of these registered features in the PLC changes, the Feature Provider calls the OnFeatureChanged method of the application. You must implement the OnFeatureChanged method in your application to respond to PLC events when registered features change.

The IFeatureCallback interfaces also allows an application to receive callback notification in the event the connection to the PLC is lost. When the Feature Provider detects a loss of connection, it calls the OnPLCDisconnect method of the application if the application has registered for connection check. You must implement the OnPLCDisconnect method in your application with logic to respond to a loss of PLC connection.

To include the IFeatureCallback interface in your application, follow the instructions for your development environment.

Visual Basic 6.0 and Visual Basic .NET

To include the IFeatureCallback interface in a Visual Basic application, include the following line at the beginning of the code for your form:

```
Implements S7WCUPIntLib.IFeatureCallback
```

Visual C++ 6.0 and Visual C++ .NET

To include the IFeatureCallback interface in a Visual C++ application, follow these steps:

1. Include code similar to the following code at the bottom of the .idl file for your project:

```
#include "FeatureCallback.idl"
[
      uuid(8FDA52F5-A330-409D-9040-B9F95DA3B4A1),
      helpstring("DemoDlg Class")
]
coclass DemoDlg
{
      [default] interface IFeatureCallback;
};
```

2. Include the following code after your #include statements in the main .h file for your project:

```
_COM_SMARTPTR_TYPEDEF(IFeatureCallback, __uuidof(IFeatureCallback));
```

3. Include the following code in the class declaration in the main .h file for your project:

```
public IDispatchImpl<IFeatureCallback, &IID_IFeatureCallback, &LIBID_DEMOLib>
```

4. Include the following declarations for the IFeatureCallback interface methods in the main .h file for your project:

```
// IFeatureCallback interface methods
STDMETHOD(OnFeatureChanged)(BSTR FeatureName, VARIANT Context, long
NotificationID, VARIANT AttributeNames, VARIANT AttributeValues);

STDMETHOD(OnPLCDisconnect)(long ErrorID);
```

Visual C# .NET

Include the following lines of code in your application:

```
using S7WCUPIntLib;
using FEATUREPROVIDERLib;
public class Form1 : System.Windows.Forms.Form, IFeatureCallback
```

Borland Delphi 7

Include the following line of code as a type declaration in your main program file:

```
type TDemoDlg = class(TForm, IFeatureCallback)
```

Including the Feature and Attribute Definitions

The Controller Management Interface supports a set of specific features and attributes. Some features and attributes apply to all types of PLCs; some are specific for a particular type of PLC. You must include definitions of these features in your application. The installation CD contains the feature and attribute definition files that you need.

To include the feature and attribute definitions, follow the instructions for your development environment.

Visual Basic 6.0

To include the feature definitions in a Visual Basic application, add the file Feature.bas as a module of your project. To add it, follow these steps:

1. Right-click on the project name in the project explorer window.
2. Select the menu command **Add > Module**.
3. Select the Existing tab and navigate to the directory that contains Feature.bas.
4. Double-click Feature.bas to add it to your project.

Visual C++ 6.0 and Visual C++ .NET

To include the feature definitions in a Visual C++ application, follow these steps:

1. Insert the following two lines in one of your header files, for example, StdAfx.h:

   ```
   #include "strdef.h"
   #include "featureStrDefine.h"
   ```

   By default, these files are located in the ...\Program Files\Siemens\WinAC\ODK\Include directory.

2. Provide a path to featureStrDefine.h and strdef.h in your project settings, following these steps based on your development environment:

   Visual C++ 6.0:

   1. Select the **Project > Settings** menu command.
   2. Select the C/C++ tab and category Preprocessor.
   3. Enter the relative path to featureStrDefine.h and strdef.h in the "Additional include directories" field.

   Visual C++ .NET:

   1. Select the **Project > Properties** menu command.
   2. Select the C/C++ tab and category General.
   3. Enter the relative path to featureStrDefine.h and strdef.h in the "Additional include directories" field.

Visual C# .NET

To include the feature definitions in a C# application, follow these steps:

1. Right-click on the project name in the project explorer window.
2. Select the menu command **Add > Existing Item**.
3. Navigate to the directory that contains to Feature.cs.
4. Double-click Feature.cs to add it to your project.

Borland Delphi 7

To include the feature definitions in a Borland Delphi 7 application, follow these steps:

1. Include the S7UP_Feature.pas file in your project by including this code in your main program file:

   ```
   interface

   uses  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
   Dialogs,  ComObj, S7UP_Interfaces, S7UP_FeatureProvider, S7UP_Feature, StdCtrls;
   ```

2. Provide a path to the S7UP_Feature.pas file, following these steps:
   a. Select the **Project > Options** menu command.
   b. Select the Directories/Conditionals tab.
   c. Enter the relative path to S7UP_Feature.pas in the "Search Path" field. By default, this file is in the ...\Program Files\Siemens\WinAC\ODK\Examples\Support_Delphi directory.

Creating an Instance of the Feature Provider

The Feature Provider COM object is the channel for communications between your application and the PLC. You must create an instance of the Feature Provider in order to use the Controller Management Interface functions for reading and writing controller information.

To create an instance of the Feature Provider, follow the instructions for your development environment.

Visual Basic 6.0

To create the Feature Provider in a Visual Basic application,

1. Declare a variable of the type IPLC, for example:

```
Private m_pIPlc As IPLC
```

2. Set this variable to a new instance of the feature provider, in one of the first statements that your application executes:

```
Set m_pIPlc = New PLC
```

Visual Basic .NET

To create the Feature Provider in a Visual Basic application,

1. Declare a variable of the type IPLC from the S7 WinAC Unified Panel Interfaces 1.0 Type Library, for example:

```
Private m_pIPlc As S7WCUPIntLib.IPLC
```

2. Set this variable to a new instance of the feature provider, in one of the first statements that your application executes:

```
Set m_pIPlc = New FEATUREPROVIDERLib.PLC
```

Visual C++ 6.0 and Visual C++ .NET

To create the Feature Provider in a Visual C++ application,

1. Declare a variable of the type IPLC, for example:

```
IPLCPtr m_pIPlc;
```

2. Set this variable to a new instance of the feature provider in the class constructor:

```
m_pIPlc.CreateInstance(__uuidof (PLC));
```

Visual C# .NET

To create the Feature Provider in a C# application,

1. Declare a variable of the type IPLC, for example:

```
private IPLC      m_pIPlc = null;
```

2. Set this variable to a new instance of the feature provider:

```
m_pIPlc = new PLCClass();
```

Borland Delphi 7

To create the Feature Provider in a Borland Delphi 7 application,

1. Declare a variable of the type IPLC, for example:

```
pIPLC: IPLC;
```

2. Set this variable to a new instance of the feature provider:

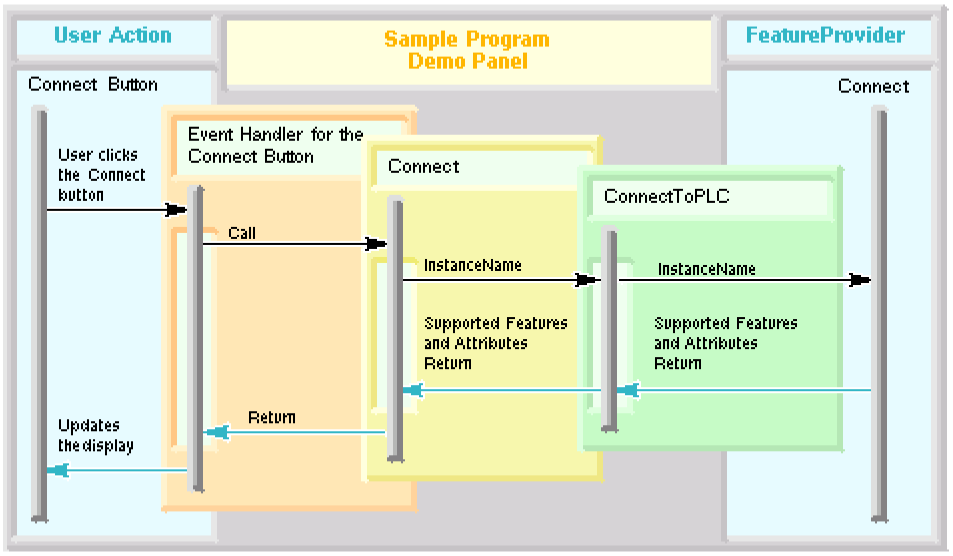
```
pIPlc = CoPLC.Create;
```

Connecting to a PLC

The WinAC Controller Management Interface provides the Connect method for connecting to any WinAC controller that is accessible on your local computer. When your application software successfully connects to a controller, the Feature Provider returns the complete set of features and attributes that the controller supports.

Note: The WinAC Controller Management Interface also provides the Browse method for finding PLCs that are accessible from your computer. The Demo_Panel sample programs use the Browse method to verify that the instance name is valid before connecting.

Each of the Demo_Panel sample programs calls an event handler when the user clicks the Connect button on the Demo dialog. The event handler method calls the internal Connect method within the application. (This is not the Connect method of the Controller Management Interface.) The application's Connect method calls the method ConnectToPlc, which calls the Connect method of the Controller Management Interface. The following diagram shows the connect sequence:


User Action
Sample Program Demo Panel
FeatureProvider
Connect Button
Connect
User clicks the Connect button
Event Handler for the Connect Button
Connect
ConnectToPLC
Call
InstanceName
InstanceName
Supported Features and Attributes Return
Supported Features and Attributes Return
Return
Updates the display

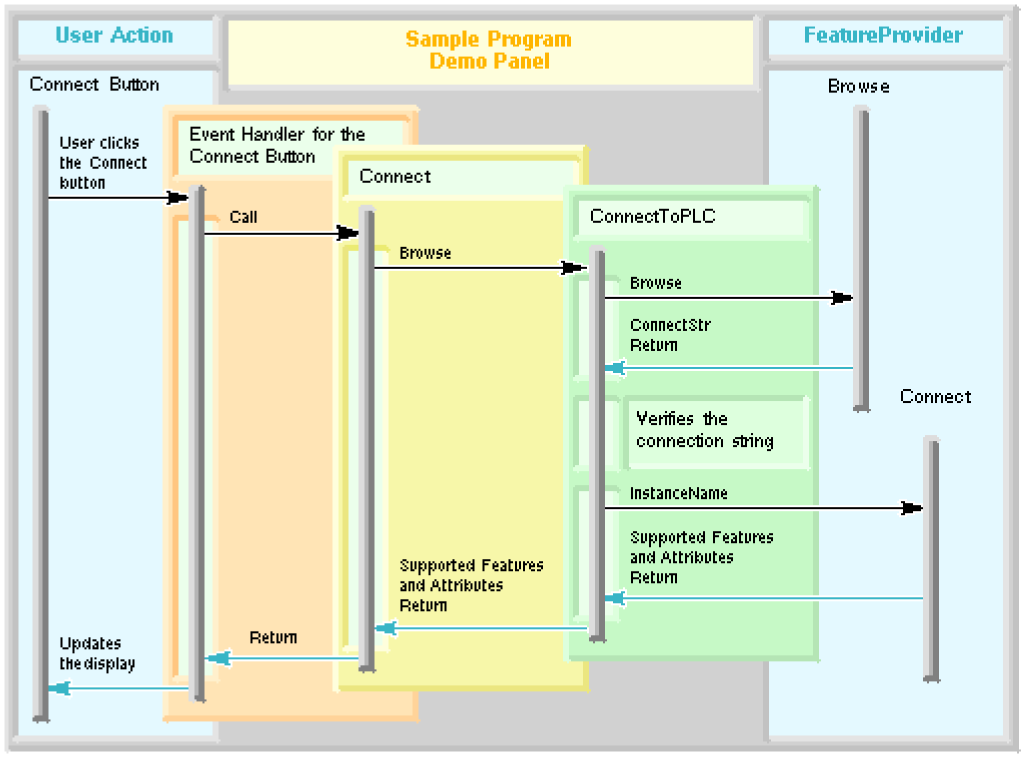
To display the software for connecting a Demo_Panel sample program to a controller, access this page in the online help.

Finding Available PLCs

The WinAC Controller Management Interface includes a Browse method that enables you to find the names of all PLCs accessible from your computer. You can use the Browse method to provide a list to the user of controllers available for connecting to your application. Another use of the Browse method is to verify that a controller name entered by the user is accessible from the computer running the application. The Demo_Panel sample programs use the Browse method for this purpose.

Each of the Demo_Panel sample programs displays a panel dialog with a connect button and and instance name input field. The instance name must be the name of an available PLC executing on the user's computer and correspond to an entry in the Station Configuration Editor, for example, "WinLC RTX".

When the user enters an instance name and clicks the Connect button, the Demo_Panel program calls the event handler for the Connect button with the instance name shown on the Demo dialog. The event handler calls the internal Connect method within the application. (This is not the Connect method of the Controller Management Interface.) The application's Connect method calls the method ConnectToPlc, which calls the internal Browse method within the application. (This is not the Browse method of the Controller Management Interface.) The internal Browse method declares and initializes variables and then calls the Browse method of the Controller Management Interface. The Browse method returns an array of connection names of PLCs that are currently available on the user's computer. ConnectToPLC then verifies that the instance name displayed on the dialog matches one of the controller strings returned by the Browse method. If the instance name does match a controller string, the ConnectToPLC method proceeds with connecting to the named controller. The controller must already be running for the ConnectToPLC method to succeed. The following diagram shows the Browse sequence as used in the Demo_Panel sample programs:


User Action
Sample Program Demo Panel
FeatureProvider
Connect Button
User clicks the Connect button
Event Handler for the Connect Button
Call
Connect
Browse
ConnectToPLC
Browse
Browse
ConnectStr Return
Connect
Verifies the connection string
InstanceName
Supported Features and Attributes Return
Supported Features and Attributes Return
Return
Updates the display

To display the Demo_Panel sample program software for browsing to find a list of available controllers, access this page in the online help.

Reading and Writing Attributes of PLC Features

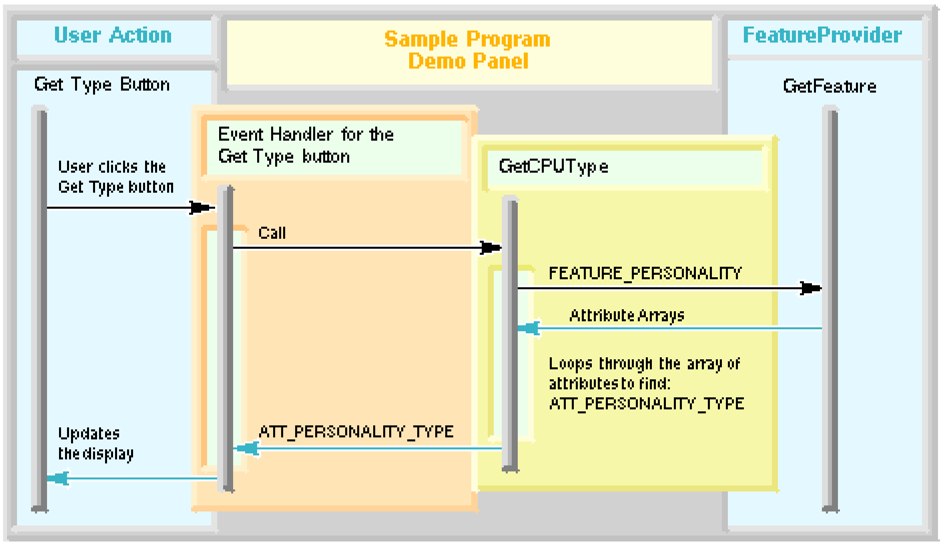
The WinAC Controller Management Interface provides methods for reading values from the controller and for writing values to the controller. Each controller defines a set of features that it supports, and each feature consists of a set of attributes. Some of these attribute values are available for reading and writing, others just for reading. The Demo_Panel sample programs contain examples of reading and writing feature attributes.

Reading Attribute Values

The Controller Management Interface provides two ways to read attribute values from the controller: getting the attribute values of a feature directly using the GetFeature method, or getting the attribute values of a feature whenever any attribute of that feature changes, as described in Responding to PLC Events.

Your application software can call the GetFeature method of the IFeature interface any time that it needs a specific attribute's value. The calling program provides the name of a feature as a parameter. The GetFeature method returns the names and values of all of the attributes that belong to that feature as well as an error return value. The calling program then can loop through the array of returned attribute values to find the particular attribute of interest.

Each of the Demo_Panel sample programs uses GetFeature in this way to retrieve the CPU personality (PLC type). When the user clicks the "Get Type" button on the control panel, the Demo_Panel program calls the event handler for the "Get Type" button, which calls the method GetCPUType. GetCPUType calls the Controller Management Interface method GetFeature for the FEATURE_PERSONALITY feature. GetFeature returns the attributes for that feature to GetCPUType. GetCPUType then loops through the attributes of the PLC personality feature and returns the attribute ATT_PERSONALITY_TYPE to OnGetType. OnGetType then updates the control panel display with the PLC type. The following diagram shows this sequence:

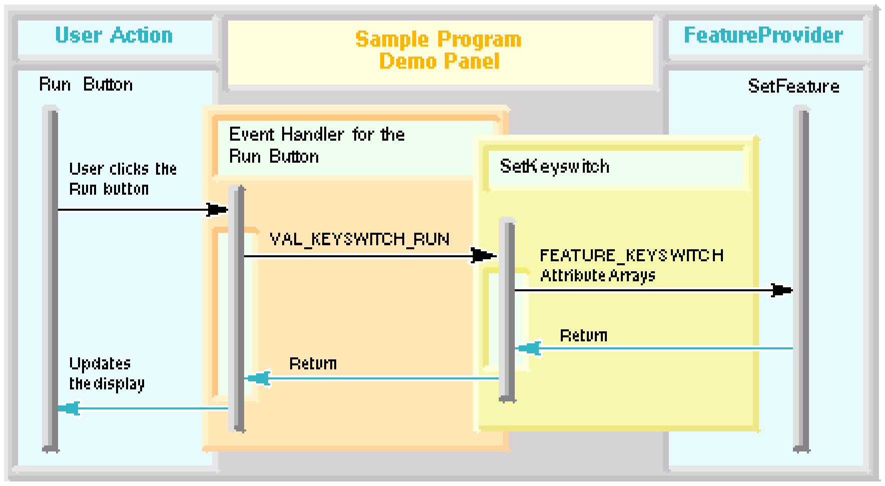

User Action
Sample Program Demo Panel
FeatureProvider
Get Type Button
GetFeature
Event Handler for the Get Type button
GetCPUType
User clicks the Get Type button
Call
FEATURE_PERSONALITY
Attribute Arrays
Loops through the array of attributes to find: ATT_PERSONALITY_TYPE
ATT_PERSONALITY_TYPE
Updates the display

To display code for getting a feature in a sample program, access this page in the online help. You can find complete code listings in the Reference section.

Writing Attribute Values

The Controller Management Interface provides the method SetFeature to write the value of a feature attribute. Your code can call SetFeature according to the application requirements, for example, when the user clicks a button, or in OnFeatureChanged when a registered feature changes.

Each of the Demo_Panel sample programs uses SetFeature to set the keyswitch position on the control panel when the user clicks the Run, RunP, Stop, or MRES button. The event handlers for the buttons call method SetKeyswitch of the Demo_Panel program. SetKeyswitch sets up the attribute arrays for FEATURE_KEYSWITCH and assigns the attribute for ATT_KEYSWITCH to the value corresponding to the button that the user pressed, for example, VAL_KEYSWITCH_RUN. SetKeyswitch then calls the Controller Management Interface method SetFeature to actually make the value change in the controller. The following diagram shows this sequence for setting the mode to RUN. The same sequence is used for each of the other attribute values that are set by the Demo_Panel program:



To display code for setting a feature in a sample program, access this page in the online help. You can find complete code listings in the Reference section.

Responding to PLC Events

You can program your application to read controller features whenever they change, and to take whatever action those changes require.

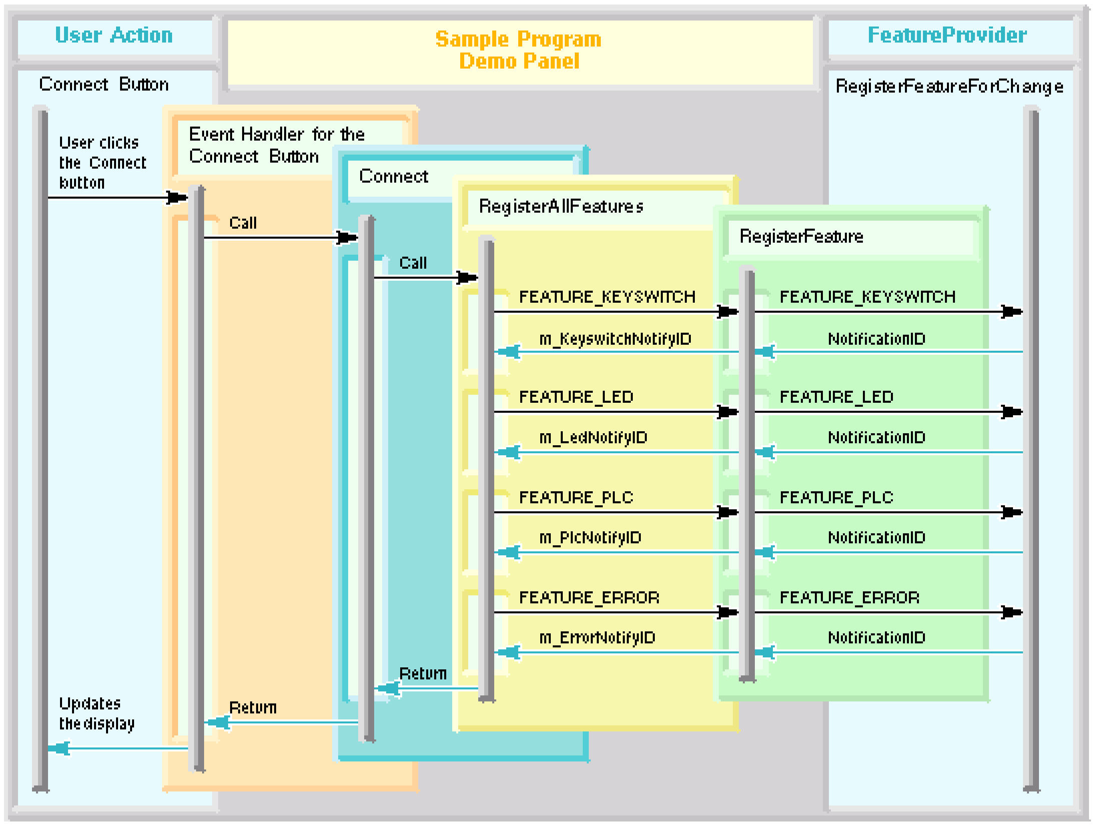
To get the values of controller features whenever they change, program your software to register one or more features for change notification. When your program registers a feature using the RegisterFeatureForChange method, it receives a callback event whenever any value of any attribute in that feature changes. The callback is a call to the OnFeatureChanged method in your application, which is a method of the IFeatureCallback interface that you must implement. The parameters sent to OnFeatureChanged are defined by the Controller Management Interface and include the values of all attributes for that feature.

Registering for and Responding to Feature Changes

Each of the Demo_Panel sample programs uses the methods for feature change notification. The design of these programs is the same; the differences are in the programming language and environment. For simplicity, this topic refers to Demo_Panel as the sample program, rather than each program by name.

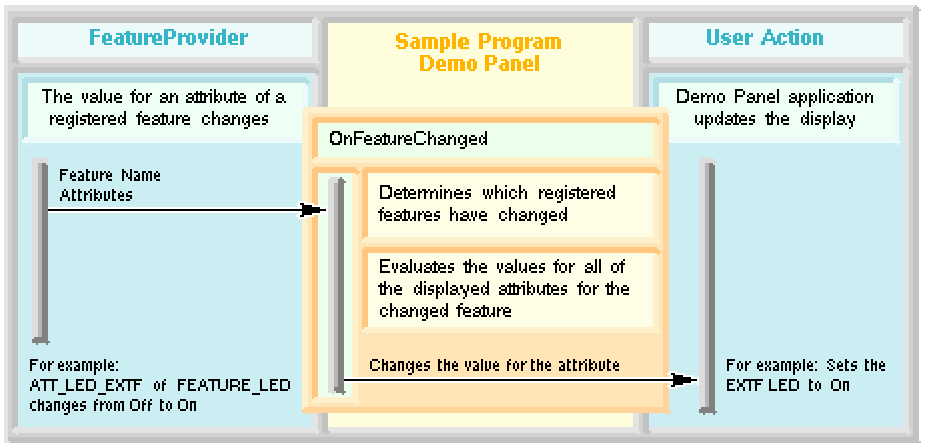
The Demo_Panel sample program connects to a PLC and then registers several features for change notification: FEATURE_KEYSWITCH, FEATURE_LED, FEATURE_PLC, and FEATURE_ERROR. The method RegisterAllFeatures calls the method RegisterFeature four times to register these four features. The method RegisterFeature is a wrapper method that accepts a FeatureName parameter, calls RegisterFeatureForChange, and returns a notification ID for the feature.

The following diagram shows the structure of feature registration in Demo_Panel:



To display code for registering features, access this page in the online help. You can find complete code listings in the Reference section.

The OnFeatureChanged method in Demo_Panel gets called whenever one of these registered features has a change in an attribute value. The parameters to OnFeatureChanged provide the feature name of the feature that changed and an array of that feature's attributes and their values. In the Demo_Panel program, the OnFeatureChanged method checks to see which of the four registered features changed. When it identifies the feature that changed, it loops through the array of attribute names and values provided by the controller. The code then sets variables that correspond to the current attribute values. For example, if the attribute ATT_LED_EXTF for feature FEATURE_LED is on, OnFeatureChanged selects the checkbox on the Demo_Panel dialog that corresponds to the EXTF (External Fault) LED. The following diagram shows the Demo_Panel program design for handling a change in one of the control panel LEDs, the EXTF indicator:


FeatureProvider
Sample Program Demo Panel
User Action
The value for an attribute of a registered feature changes
Feature Name Attributes
For example: ATT_LED_EXTF of FEATURE_LED changes from Off to On
OnFeatureChanged
Determines which registered features have changed
Evaluates the values for all of the displayed attributes for the changed feature
Changes the value for the attribute
Demo Panel application updates the display
For example: Sets the EXTF LED to On

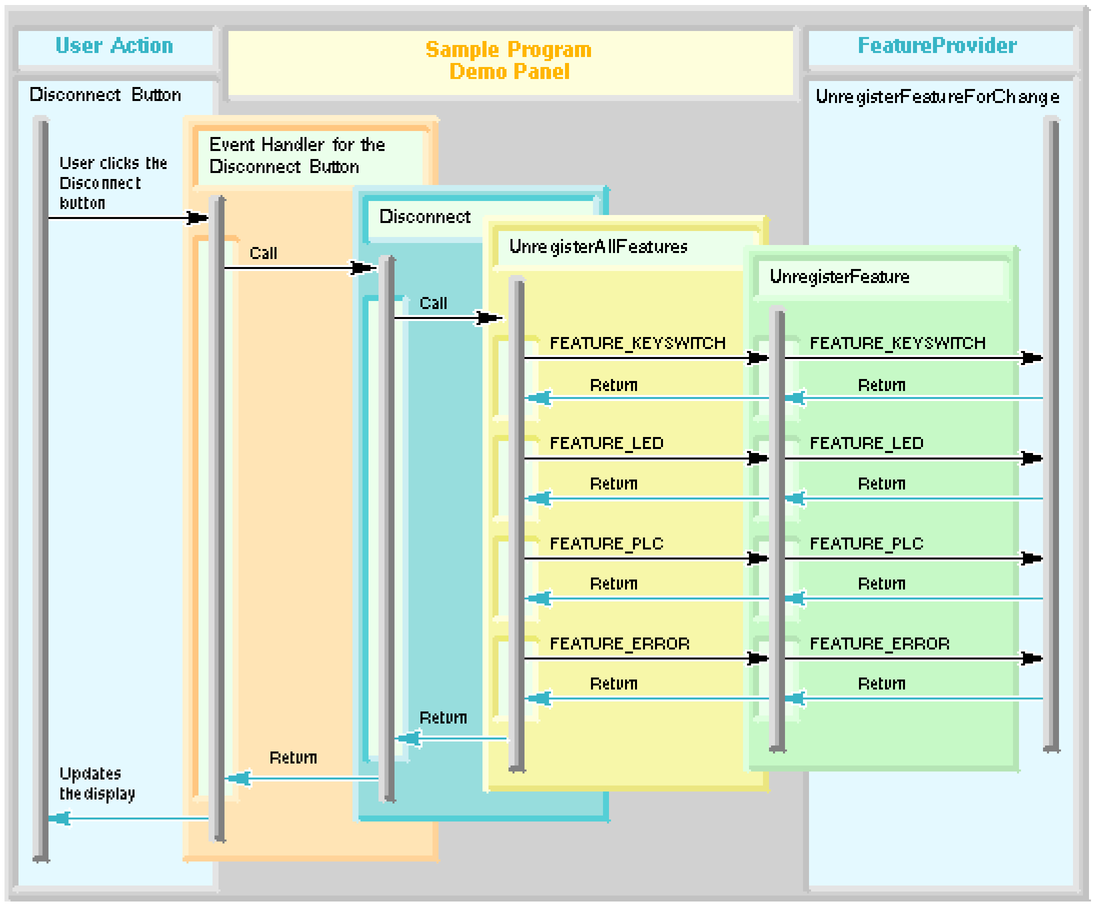
To display code for responding to feature changes, access this page in the online help. You can find complete code listings in the Reference section.

Unregistering Features

If your application no longer needs a callback when a particular feature changes, you can unregister that feature as one requiring change notification. You can use the method UnregisterFeatureForChange to unregister a single feature.

Also, your program termination code should unregister all features that the program registered for change notification. The sample program Demo_Panel contains a method UnregisterAllFeatures that calls a wrapper method UnregisterFeature for each of the four registered features. The wrapper method UnregisterFeature calls the Controller Management Interface method UnregisterFeatureForChange for each feature. The sample program Demo_Panel calls the method UnregisterAllFeatures from its Disconnect method, which is called from the OnConnectBtn event handler when the user clicks the Disconnect button.

The following diagram shows the sequence in the Demo_Panel programs for unregistering features:

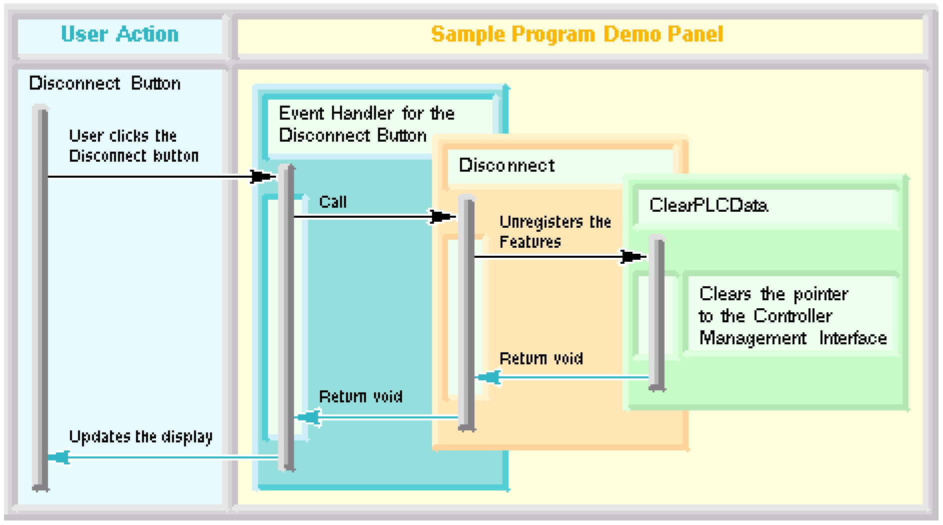


To display code for unregistering features, access this page in the online help. You can find complete code listings in the Reference section.

Disconnecting from a PLC

To disconnect your application from a controller, your software must release the controller management interface. Prior to releasing the controller management interface, your software should unregister any features that it has registered for change notification. After your software releases the controller management interface, it can update the display to reflect a disconnected state.

The dialog for the Demo_Panel sample programs contains a button that toggles between "Connect" and "Disconnect" button. The event handler for this button responds to two key events for this button: Connect when the program is not connected to a controller, and Disconnect when the program is connected to a controller. When the user clicks Disconnect, the OnConnectBtn event handler calls an internal Disconnect method. The Disconnect method unregisters all registered features, and then calls an internal ClearPLCData function. Depending on the programming environment, ClearPLCData may need to get a pointer to the controller management interface before it can release it. ClearPLCData then releases the controller management interface from the application, breaking the connection to the Feature Provider. The diagram below shows the Disconnect sequence in the Demo_Panel sample programs:


User Action
Sample Program Demo Panel
Disconnect Button
User clicks the Disconnect button
Event Handler for the Disconnect Button
Call
Disconnect
Unregisters the Features
ClearPLCData
Clears the pointer to the Controller Management Interface
Return void
Return void
Updates the display

To display code for disconnecting from a controller, access this page in the online help. You can find complete code listings in the Reference section.

Responding to Loss of PLC Connection

The WinAC Controller Management Interface can provide a callback to your application in the event the connection to the PLC is lost. You must register for this callback using the method RegisterForConnectionCheck. In the event the Feature Provider loses communication with the PLC, it generates a call to the method OnPLCDisconnect in your application. To unregister for this notification, your application can call UnregisterForConnectionCheck.

You program the OnPLCDisconnect method within your application with code to handle a loss of communication with the PLC. For example, your application could display a message to the user, deactivate controls on the dialog, close dialogs, or whatever is appropriate.

Notice
Shutting down the PLC does not cause a call to OnPLCDisconnect. A PLC shutdown is a normal action for the PLC and is not a loss in connection. Your application can still communicate with a shutdown PLC and be notified, for example, when the PLC is started.

The Controller Management Interface does not call OnPLCDisconnect when the user disconnects the application from the PLC. The communication connection is still available in this situation for the user to reconnect the application to the PLC. The Controller Management Interface only calls OnPLCDisconnect when the Controller Management Interface itself cannot communicate with the PLC, and thus cannot provide a connection interface between the PLC and the application.

The Demo_Panel sample programs call RegisterForConnectionCheck in the RegisterAllFeatures method. (The RegisterAllFeatures method also registers specific features to enable the application to respond to PLC events.)

The OnPLCDisconnect method in the Demo_Panel sample programs calls an internal ConnectionLost method. This method releases the controller interface, deactivates all controls on the dialog, and clears the PLC type display field. It changes the text of the Connect/Disconnect button to "Connect" but deactivates this button. The user must shut down the Demo_Panel application, correct any problems, and restart the Demo_Panel application in order to reestablish a connection with a controller.

The Demo_Panel sample programs call UnregisterForConnectionCheck in the UnregisterAllFeatures method.

To display code in one of the Demo_Panel sample programs for handling loss of connection, access this page in the online help. You can find complete code listings in the Reference section.

Additional Sample Programs

In addition to the Demo Panel sample programs, WinAC ODK 4.1 includes an additional sample program in each programming environment that demonstrates how to archive and restore your PLC with the CMI MemoryCardFile feature. WinAC ODK 4.1 also includes a sample program in C++ only that demonstrates how to save diagnostic information for the controller.

The sample programs are in the ...\Program Files\Siemens\WinAC\ODK\Examples\CMI folder. The supported development environments and folders containing sample programs for those environments are listed below.

Programming Development Environment	Sample Project Folder
Visual Basic 6.0	CMI_ArchiveRestore_Vb
Visual Basic .NET	CMI_NET_ArchiveRestore_VB
Visual C++ 6.0	CMI_ArchiveRestore_Cpp CMI_Diag_Cpp
Visual C++ .NET	CMI_ArchiveRestore_Cpp CMI_Diag_Cpp
Visual C# .NET (C sharp)	CMI_ArchiveRestore_Cs
Borland Delphi 7	CMI_ArchiveRestore_Delphi

Note
C++ source files do not differ between Visual C++ 6.0 and Visual C++ .NET. The sample project folders for C++ have a project workspace for Visual C++ 6.0, a solution workspace for Visual C++ .NET and a set of common source files.

Programming Tips and Error Handling

In your CMI application, verify the results of each CMI method call before using data from the call in subsequent program logic. Verify that parameters for the CMI methods are suitable for the particular task you intend it to perform.

Warning
Do not rely on the pErrorID return value from CMI methods, or the returned HRESULT value for C++ methods. Although a non-zero value for pErrorID does indicate that the CMI method was unsuccessful, the CMI methods can return a pErrorID value of 0 and a return value of 0 even when errors exist. The return value pErrorID is for future use; currently, it is not a reliable indicator of the outcome of a method call.

CMI programs that rely solely on the pErrorID value or HRESULT value to indicate the success of method calls can misrepresent information in the PLC, permitting conditions that could result in equipment damage or injury to personnel.

Verify your expected results from the CMI methods by checking the input and output data, and by responding to a loss of PLC connection.

Verifying a Connection to a PLC

When your CMI application calls the Connect method to connect to a PLC, the Feature Provider returns a complete set of features and attributes for that PLC when the Connect call succeeds.

Program your CMI application to check that the array of feature names and the array of attribute names is not empty. An empty feature and attribute array means that the Connect call was unsuccessful, probably due to an invalild ConnectString parameter.

You can also call the Browse method prior to calling the Connect method, and verify that the ConnectString parameter for the Connect method corresponds to an entry in the pConnectStrings array returned by the Browse method. The Connect method can only connect to PLCs that the Browse method identifies.

Ensuring that Feature and Attribute Names are Valid

Before your CMI application calls GetFeature or SetFeature to read or write attribute values for a particular feature, verify that the feature name and the attribute names are valid for the PLC to which your application is connected.

Your application must successfully connect to a PLC before it can get features or set features. The Connect method returns arrays of features and attributes for the PLC when the connection attempt succeeds. Before calling GetFeature or SetFeature, verify that the parameters containing the feature name and the attribute names correspond to values in the arrays returned by the Connect method.

Handling a Loss of PLC Connection

Your CMI application most likely needs to take some action if the connection to the PLC is lost. You can, for example, program buttons and fields on your user interface to be desensitized when there is no valid PLC connection. If your application is performing data collection, or some other task that does not involve a user interface, you can also program logic to handle a connection loss according to your specific requirements.

To program an application to respond to a loss of PLC connection, register for connection check in your application and program an OnPLCDisconnect method to perform tasks that are appropriate in the event of a PLC connection loss.

Following SetFeature Calls with GetFeature Calls

You can follow SetFeature calls in your application with calls to GetFeature to verify that the attribute values that you intended to set are actually set in the PLC.

For example, if your application writes a new value for the Keyswitch feature, your application can then read the Keyswitch value that it just set. You can also program your application to read the Run and Stop attributes of the LED feature to verify that your keyswitch setting did in fact change the operating mode correctly.

Reference

Method Declarations

Visual Basic Method Declarations

The Visual Basic declarations as shown in the object browser for the IPLC, IFeatureInterface, and IFeatureCallback methods are listed below:

Browse

```
Sub Browse(
   pConnectStrings,
   pStartInfos,
   pErrorID As Long)
```

Connect

```
Sub Connect(
   ConnectString As String,
   pIFeature As IFeature,
   pFeatureNames,
   pAttributeNamesArray,
   pErrorID As Long)
```

GetFeature

```
Sub GetFeature(
   FeatureName As String,
   pAttributeNames,
   pAttributeValues,
   pErrorID As Long)
```

OnFeatureChanged

```
Sub OnFeatureChanged(
   FeatureName As String,
   Context,
   NotificationID As Long,
   AttributeNames,
   AttributeValues)
```

OnPLCDisconnect

```
Sub OnPLCDisconnect(ErrorID As Long)
```

RegisterFeatureForChange

```
Sub RegisterFeatureForChange(
   pCallback As IFeatureCallback,
   FeatureName As String,
   Context,
   pNotificationID As Long,
   pErrorID As Long)
```

RegisterForConnectionCheck

```
Sub RegisterForConnectionCheck(
   pCallback As IFeatureCallback,
   pNotificationID As Long,
   pErrorID As Long)
```

SetFeature

```
Sub SetFeature(
   FeatureName As String,
   AttributeNames,
   AttributeValues,
  pErrorID As Long)
```

UnregisterFeatureForChange

```
Sub UnregisterFeatureForChange(
   NotificationID As Long,
   pErrorID As Long)
```

UnregisterForConnectionCheck

```
Sub UnregisterForConnectionCheck(
   NotificationID As Long,
   pErrorID As Long)
```

Visual Basic .NET Method Declarations

The Visual Basic .NET declarations as shown in the object browser for the IPLC, IFeatureInterface, and IFeatureCallback methods are listed below:

Browse

```
Public Sub Browse(
   ByRef pConnectStrings As Object,
   ByRef pStartInfos As Object,
   ByRef pErrorID As Integer)
```

Connect

```
Public Sub Connect(
   ByVal ConnectString As String,
   ByRef pIFeature As S7WCUPIntLib.IFeature,
   ByRef pFeatureNames As Object,
   ByRef pAttributeNamesArray As Object,
   ByRef pErrorID As Integer)
```

GetFeature

```
Public Sub GetFeature(
   ByVal FeatureName As String,
   ByRef pAttributeNames As Object,
   ByRef pAttributeValues As Object,
   ByRef pErrorID As Integer)
```

OnFeatureChanged

```
Public Sub OnFeatureChanged(
   FeatureName As String,
   Context,
   NotificationID As Integer,
   AttributeNames,
   AttributeValues)
```

OnPLCDisconnect

```
Public Sub OnPLCDisconnect(ErrorID As Integer)
```

RegisterFeatureForChange

```
Public Sub RegisterFeatureForChange(
   ByVal pCallback As S7VCUPIntLib.IFeatureCallback,
   ByVal FeatureName As String,
   ByVal ContextAs Object,
   ByRef pNotificationID As Integer,
   ByRef pErrorID As Integer)
```

RegisterForConnectionCheck

```
Public Sub RegisterForConnectionCheck(
   ByVal pCallback As S7WCUPIntLib.IFeatureCallback,
   ByRef pNotificationID As Integer,
   ByRef pErrorID As Integer)
```

SetFeature

```
Public Sub SetFeature(
   ByVal FeatureName As String,
   ByVal AttributeNames As Object,
   ByVal AttributeValues As Object,
 ByRef pErrorID As Integer)
```

UnregisterFeatureForChange

```
Public Sub UnregisterFeatureForChange(
   ByVal NotificationID As Integer,
   ByRef pErrorID As Integer)
```

UnregisterForConnectionCheck

```
Public Sub UnregisterForConnectionCheck(
   ByVal NotificationID As Integer,
   ByRef pErrorID As Integer)
```

Visual C++ 6.0 and .NET Method Declarations

The Visual C++ declarations for the IPLC, IFeatureInterface, and IFeatureCallback methods are listed below. These declarations are applicable for both Visual C++ 6.0 and Visual C++ .NET.

Browse

```
HRESULT Browse(
   [in,out] VARIANT* pConnectStrings,
   [in,out] VARIANT* pStartInfos,
   [in,out] long* pErrorID);
```

Connect

```
HRESULT Connect(
   [in] BSTR ConnectString,
   [in,out] IFeature** pIFeature,
   [in,out] VARIANT* pFeatureNames,
   [in,out] VARIANT* pAttributeNamesArray,
   [in,out] long* pErrorID);
```

GetFeature

```
HRESULT GetFeature(
   [in] BSTR FeatureName,
   [in,out] VARIANT* pAttributeNames,
   [in,out] VARIANT* pAttributeValues,
   [in,out] long* pErrorID);
```

OnFeatureChanged

```
HRESULT OnFeatureChanged(
   [in] BSTR FeatureName,
   [in] VARIANT Context,
   [in] long NotificationID,
   [in] VARIANT AttributeNames,
   [in] VARIANT AttributeValues);
```

OnPLCDisconnect

```
HRESULT OnPLCDisconnect([in] long ErrorID);
```

RegisterFeatureForChange

```
HRESULT RegisterFeatureForChange(
   [in] IFeatureCallback* pCallback,
   [in] BSTR FeatureName,
   [in] VARIANT Context,
   [in,out] long* pNotificationID,
   [in,out] long* pErrorID);
```

RegisterForConnectionCheck

```
HRESULT RegisterForConnectionCheck(
   [in] IFeatureCallback* pCallback,
   [in,out] long* pNotificationID,
   [in,out] long* pErrorID);
```

SetFeature

```
HRESULT SetFeature(
   [in] BSTR FeatureName,
   [in] VARIANT AttributeNames,
   [in] VARIANT AttributeValues,
   [in,out] long* pErrorID);
```

UnregisterFeatureForChange

```
HRESULT UnregisterFeatureForChange(
   [in] long NotificationID,
   [in,out] long* pErrorID);
```

UnregisterForConnectionCheck

```
HRESULT UnregisterForConnectionCheck(
   [in] long NotificationID,
   [in,out] long* pErrorID);
```

C# Method Declarations

The C# declarations for the IPLC, IFeatureInterface, and IFeatureCallback methods are listed below:

Browse

```
instance void  Browse(
   [in][out] object&  marshal( struct) pConnectStrings,
   [in][out] object&  marshal( struct) pStartInfos,
   [in][out] int32& pErrorID) runtime managed internalcall
```

Connect

```
instance void  Connect(
   [in] string  marshal( bstr) ConnectString,
   [in][out] class S7UnifiedPanel.IFeature&  marshal( interface)
pIFeature,
   [in][out] object&  marshal( struct) pFeatureNames,
   [in][out] object&  marshal( struct) pAttributeNamesArray,
   [in][out] int32& pErrorID) runtime managed internalcall
```

GetFeature

```
instance void  GetFeature(
   [in] string  marshal( bstr) FeatureName,
   [in][out] object&  marshal( struct) pAttributeNames,
   [in][out] object&  marshal( struct) pAttributeValues,
   [in][out] int32& pErrorID) runtime managed internalcall
```

OnFeatureChanged

```
instance void  OnFeatureChanged(
   [in] string  marshal( bstr) FeatureName,
   [in] object  marshal( struct) Context,
   [in] int32   NotificationID,
   [in] object  marshal( struct) AttributeNames,
   [in] object  marshal( struct) AttributeValues) runtime managed
internalcall
```

OnPLCDisconnect

```
instance void OnPLCDisconnect(
   [in] int32 ErrorID) runtime managed internalcall
```

RegisterFeatureForChange

```
instance void  RegisterFeatureForChange(
   [in] class    S7UnifiedPanel.IFeatureCallback  marshal( interface)
pCallback,
   [in] string marshal( bstr) FeatureName,
   [in] object marshal( struct) Context,
   [in][out] int32& pNotificationID,
   [in][out] int32& pErrorID) runtime managed internalcall
```

RegisterForConnectionCheck

```
instance void RegisterForConnectionCheck(
   [in] class S7UnifiedPanel.IFeatureCallback marshal( interface)
pCallback,
   [in][out] int32& pNotificationID,
   [in][out] int32& pErrorID) runtime managed internalcall
```

SetFeature

```
instance void  SetFeature(
   [in] string  marshal( bstr) FeatureName,
   [in] object  marshal( struct) AttributeNames,
   [in] object  marshal( struct) AttributeValues,
   [in][out] int32& pErrorID) runtime managed internalcall
```

UnregisterFeatureForChange

```
instance void  UnregisterFeatureForChange(
   [in] int32 NotificationID,
   [in][out] int32& pErrorID) runtime managed internalcall
```

UnregisterForConnectionCheck

```
instance void  UnregisterForConnectionCheck(
   [in] int32 NotificationID,
   [in][out] int32& pErrorID) runtime managed internalcall
```

Borland Delphi 7 Method Declarations

The Borland Delphi 7 declarations for the IPLC, IFeatureInterface, and IFeatureCallback methods are listed below:

Browse

```
procedure Browse(
   var pConnectStrings: OleVariant;
   var pStartInfos: OleVariant;
   var pErrorID: Integer); safecall;
```

Connect

```
procedure Connect(
   const ConnectString: WideString;
   var pIFeature: IFeature;
   var pFeatureNames: OleVariant;
   var pAttributeNamesArray: OleVariant;
   var pErrorID: Integer); safecall;
```

GetFeature

```
procedure GetFeature(
   const FeatureName: WideString;
   var pAttributeNames: OleVariant;
   var pAttributeValues: OleVariant;
   var pErrorID: Integer); safecall;
```

OnFeatureChanged

```
procedure OnFeatureChanged(
   const FeatureName: WideString; Context: OleVariant;
   NotificationID: Integer;
   AttributeNames: OleVariant;
   AttributeValues: OleVariant); safecall;
```

OnPLCDisconnect

```
procedure OnPLCDisconnect(ErrorID: Integer); safecall;
```

RegisterFeatureForChange

```
procedure RegisterFeatureForChange(
   const pCallback: IFeatureCallback;
   const FeatureName: WideString; Context: OleVariant;
   var pNotificationID: Integer;
   var pErrorID: Integer); safecall;
```

RegisterForConnectionCheck

```
procedure  RegisterForConnectionCheck(
   const pCallback: IFeatureCallback;
   var pNotificationID: Integer;
   var pErrorID: Integer); safecall;
```

SetFeature

```
procedure SetFeature(
   const FeatureName: WideString;
   AttributeNames: OleVariant;
   AttributeValues: OleVariant;
   var pErrorID: Integer); safecall;
```

UnregisterFeatureForChange

```
procedure UnregisterFeatureForChange(
   NotificationID: Integer;
   var pErrorID: Integer); safecall;
```

UnregisterForConnectionCheck

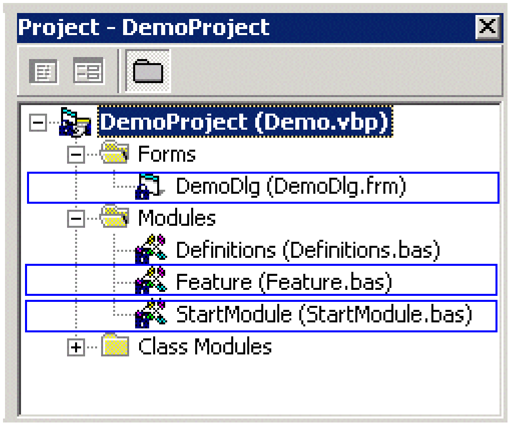
```
procedure UnregisterForConnectionCheck(
   NotificationID: Integer;
   var pErrorID: Integer); safecall;
```

Visual Basic 6.0 Sample Program

Introduction to the Visual Basic 6.0 Demo Panel Program

As described in Introducing the Sample Programs, WinAC ODK installs a Visual Basic 6.0 implementation of a control panel that can interact with a WinAC controller. This project is located in the ...\Program Files\Siemens\WinAC\ODK\Examples\CMI\CMI_Demo_Panel_VB directory.

You can click one of the indicated files in the VB Project Browser picture to access its code listing (online help only):



Forms

DemoDlg.frm

```
Option Explicit
Implements S7WCUPIntLib.IFeatureCallback

Private hr As Long
Private ErrorID As Long
Private m_pIPlc As IPLC
Private m_pIFeature As IFeature
Private m_FeatureNames As Variant
Private m_ConnectionNotifyID As Long
Private m_KeyswitchNotifyID As Long
Private m_LedNotifyID As Long
Private m_PlcNotifyID As Long
Private m_ErrorNotifyID As Long

Private Sub ActivateControls()
    'activate buttons
    m_btnRUNP.Enabled = True
    m_btnRUN.Enabled = True
    m_btnSTOP.Enabled = True
    m_btnMRES.Enabled = True
    m_btnType.Enabled = True
End Sub

Private Sub DeactivateControls()
    'uncheck all radio buttons
    m_chkON.Value = False
    m_chkBATF.Value = False
    m_chkINTF.Value = False
    m_chkEXTF.Value = False
    m_chkBUSF1.Value = False
    m_chkBUSF2.Value = False
    m_chkFRCE.Value = False
    m_chkRUN.Value = False
    m_chkSTOP.Value = False
    m_chkRUNPbtn.Value = False
    m_chkRUNbtn.Value = False
    m_chkSTOPbtn.Value = False

    'deactivate buttons
    m_btnRUNP.Enabled = False
    m_btnRUN.Enabled = False
    m_btnSTOP.Enabled = False
    m_btnMRES.Enabled = False
    m_btnType.Enabled = False
End Sub

Private Function Connect() As Long
    'init
    Dim AttributeNames As Variant

    'get the instance name
    Dim strInstanceName As String
    strInstanceName = m_edtInstanceName.Text
    'disable the instance name box
    m_edtInstanceName.Enabled = False

    'release all PLC related data
    Call Me.ClearPlcData
    'connect to the PLC
    hr = Me.ConnectToPlc(strInstanceName, m_pIFeature, m_FeatureNames,
AttributeNames)

    'if connection succeeded...
    If (SUCCEEDED(hr) = True) Then

        'register all necessary features
        hr = Me.RegisterAllFeatures()

    End If

    Connect = hr
End Function

Private Sub Disconnect()
    'unregister all features
    Call Me.UnregisterAllFeatures

    'clear all PLC related data
    Call Me.ClearPlcData

    'deactivate all controls
    Call DeactivateControls

    'clear the plc type info
    m_edtType.Text = ""

    'enable the instance name box
    m_edtInstanceName.Enabled = True
End Sub

Public Sub ConnectionLost()
    'clear the plc proxy
    Call Me.ClearPlcData

    'deactivate all controls
    Call DeactivateControls

    'change the connect button
    m_btnConnect.Caption = BTN_TXT_CONNECT

    'disable the connect button => app needs to be restartet
    m_btnConnect.Enabled = False

    'clear the plc type info
    m_edtType.Text = ""
End Sub

Private Sub m_btnConnect_Click()
    'lock the button
    m_btnConnect.Enabled = False

    'get the button text
    Dim strBtnText As String
    strBtnText = m_btnConnect.Caption

    'if we want to connect...
    If (strBtnText = BTN_TXT_CONNECT) Then

        'connect to the PLC
        hr = Connect()

        'if everything is fine...
        If (SUCCEEDED(hr) = True) Then

            'change the connect button
            m_btnConnect.Caption = BTN_TXT_DISCONNECT

        'if connection failed...
        Else

            'disconnect the PLC and clear all settings
            Call Disconnect

            'show error message box
            MsgBox ("Connecting to the PLC failed.")

        End If

    'if we want to disconnect...
    ElseIf (strBtnText = BTN_TXT_DISCONNECT) Then

        'disconnect the PLC
        Call Disconnect

        'change the connect button
        m_btnConnect.Caption = BTN_TXT_CONNECT

    End If

    'unlock the button
    m_btnConnect.Enabled = True
End Sub

Private Sub m_btnMRES_Click()
    'create message
    Dim Message As String
    Message = "The module will be reset (clear/reset).  All user data will be
deleted and all existing connections to the module will be disconnected.  Do
you really want to reset the module?"

    'double-check with the user
    If MsgBox(Message, vbYesNo + vbExclamation + vbApplicationModal, "Demo") =
vbYes Then

        'send a request to change the keyswitch to MRES
        Me.SetKeyswitch (VAL_KEYSWITCH_MRES)
    End If

    'send a request to change the keyswitch to MRES
    Me.SetKeyswitch (VAL_KEYSWITCH_MRES)
End Sub

Private Sub m_btnRUN_Click()
    'send a request to change the keyswitch to RUN
    Me.SetKeyswitch (VAL_KEYSWITCH_RUN)
End Sub

Private Sub m_btnRUNP_Click()
    'send a request to change the keyswitch to RUN-P
    Me.SetKeyswitch (VAL_KEYSWITCH_RUNP)
End Sub

Private Sub m_btnSTOP_Click()
    'send a request to change the keyswitch to STOP
    Me.SetKeyswitch (VAL_KEYSWITCH_STOP)
End Sub

Private Sub m_btnType_Click()
    'init
    Dim strPlcType As String

    'get the plc type info
    hr = Me.GetCpuType(strPlcType)

    'error
    If (FAILED(hr) = True) Then
        strPlcType = "error"
    ElseIf (hr = S_FALSE) Then
        strPlcType = "unknown"
    End If

    'display the plc type
    m_edtType.Text = strPlcType
End Sub

Private Sub Form_Load()
    'create an instance of the feature provider
    Set m_pIPlc = New PLC

    'set default instance name
    m_edtInstanceName.Text = "WinLC"

    'clear plc type info
    m_edtType.Text = ""

    'set the initial connect button text
    m_btnConnect.Caption = BTN_TXT_CONNECT

    'disable all panel controls
    Call DeactivateControls
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
    'disconnect
    Call Disconnect

    'Release the feature provider
    Set m_pIPlc = Nothing

End Sub

Public Sub IfeatureCallback_OnFeatureChanged(ByVal FeatureName As String, ByVal
Context As Variant, ByVal NotIficationID As Long, ByVal AttributeNames As
Variant, ByVal AttributeValues As Variant)
    'init
    Dim strName As String
    Dim strValue As String
    Dim i As Long

    'if it's the keyswitch feature...
    If (FeatureName = FEATURE_KEYSWITCH) Then

        'go thru all attributes...
        For i = LBound(AttributeNames) To UBound(AttributeNames)

            'get the attribute name
            strName = AttributeNames(i)

            'if it's the keyswitch attribute...
            If (strName = ATT_KEYSWITCH) Then

                'get the attribute value
                strValue = AttributeValues(i)

                'if the keyswitch is set to RUNP...
                If (strValue = VAL_KEYSWITCH_RUNP) Then
                    m_chkRUNPbtn.Value = 1
                    m_chkRUNbtn.Value = 0
                    m_chkSTOPbtn.Value = 0
                'if the keyswitch is set to RUN...
                ElseIf (strValue = VAL_KEYSWITCH_RUN) Then
                    m_chkRUNPbtn.Value = 0
                    m_chkRUNbtn.Value = 1
                    m_chkSTOPbtn.Value = 0
                'if the keyswitch is set to STOP...
                ElseIf (strValue = VAL_KEYSWITCH_STOP) Then
                    m_chkRUNPbtn.Value = 0
                    m_chkRUNbtn.Value = 0
                    m_chkSTOPbtn.Value = 1
                End If
            End If
        Next i

    'if it's the led feature...
    ElseIf (FeatureName = FEATURE_LED) Then

        'go thru all attributes...
        For i = LBound(AttributeNames) To UBound(AttributeNames)
```

```
'get the attribute name
strName = AttributeNames(i)

'if it's the power led attribute...
If (strName = ATT_LED_POWER) Then

    'get the attribute value
    strValue = AttributeValues(i)

    'if the led is off...
    If (strValue = VAL_LED_OFF) Then
        m_chkON.Value = 0
    'if it is not off it is on (we don't show blinking LEDs)
    Else
        m_chkON.Value = 1
    End If

'if it's the batf led attribute...
ElseIf (strName = ATT_LED_BATF) Then

    'get the attribute value
    strValue = AttributeValues(i)

    'if the led is off...
    If (strValue = VAL_LED_OFF) Then
        m_chkBATF.Value = 0
    'if it is not off it is on (we don't show blinking LEDs)
    Else
        m_chkBATF.Value = 1
    End If

'if it's the intf led attribute...
ElseIf (strName = ATT_LED_INTF) Then

    'get the attribute value
    strValue = AttributeValues(i)

    'if the led is off...
    If (strValue = VAL_LED_OFF) Then
        m_chkINTF.Value = 0
    'if it is not off it is on (we don't show blinking LEDs)
    Else
        m_chkINTF.Value = 1
    End If

'if it's the extf led attribute...
ElseIf (strName = ATT_LED_EXTF) Then

    'get the attribute value
    strValue = AttributeValues(i)

    'if the led is off...
    If (strValue = VAL_LED_OFF) Then
        m_chkEXTF.Value = 0
    'if it is not off it is on (we don't show blinking LEDs)
    Else
        m_chkEXTF.Value = 1
    End If

'if it's the busf1 led attribute...
ElseIf (strName = (ATT_LED_BUSF & "0")) Then

    'get the attribute value
    strValue = AttributeValues(i)

    'if the led is off...
    If (strValue = VAL_LED_OFF) Then
        m_chkBUSF1.Value = 0
    'if it is not off it is on (we don't show blinking LEDs)
    Else
        m_chkBUSF1.Value = 1
    End If

'if it's the busf2 led attribute...
ElseIf (strName = (ATT_LED_BUSF & "1")) Then

    'get the attribute value
    strValue = AttributeValues(i)

    'if the led is off...
    If (strValue = VAL_LED_OFF) Then
        m_chkBUSF2.Value = 0
    'if it is not off it is on (we don't show blinking LEDs)
    Else
        m_chkBUSF2.Value = 1
    End If

'if it's the frce led attribute...
ElseIf (strName = ATT_LED_FORCE) Then

    'get the attribute value
    strValue = AttributeValues(i)

    'if the led is off...
    If (strValue = VAL_LED_OFF) Then
        m_chkFRCE.Value = 0
    'if it is not off it is on (we don't show blinking LEDs)
    Else
        m_chkFRCE.Value = 1
    End If

'if it's the run led attribute...
ElseIf (strName = ATT_LED_RUN) Then

    'get the attribute value
    strValue = AttributeValues(i)

    'if the led is off...
    If (strValue = VAL_LED_OFF) Then
        m_chkRUN.Value = 0
    'if it is not off it is on (we don't show blinking LEDs)
    Else
        m_chkRUN.Value = 1
    End If
```

```
            'if it's the stop led attribute...
            ElseIf (strName = ATT_LED_STOP) Then

                'get the attribute value
                strValue = AttributeValues(i)

                'if the led is off...
                If (strValue = VAL_LED_OFF) Then
                    m_chkSTOP.Value = 0
                'if it is not off it is on (we don't show blinking LEDs)
                Else
                    m_chkSTOP.Value = 1
                End If
            End If
        Next i

    'if it's the plc feature...
    ElseIf (FeatureName = FEATURE_PLC) Then

        'go thru all attributes...
        For i = LBound(AttributeNames) To UBound(AttributeNames)

            'get the attribute name
            strName = AttributeNames(i)

            'if it's the plc attribute...
            If (strName = ATT_PLC) Then

                'get the attribute value
                strValue = AttributeValues(i)

                'if the PLC is created...
                If (strValue = VAL_PLC_CREATED) Then
                    Call ActivateControls
                'if the PLC not running...
                Else
                    Call DeactivateControls
                End If
            End If
        Next i

    'if it's the error feature...
    ElseIf (FeatureName = FEATURE_ERROR) Then

        'go thru all attributes...
        For i = LBound(AttributeNames) To UBound(AttributeNames)

            'get the attribute name
            strName = AttributeNames(i)

            'if it's the error id attribute...
            If (strName = ATT_ERROR_ID) Then

                'get the attribute value
                strValue = AttributeValues(i)
                'if it's an error....
                If (strValue <> PSERR_OKAY) Then

                    'create message
                    Dim strMessage As String
                    strMessage = "The FeatureProvider returned an error."

                    'show error message
                    MsgBox (strMessage)
                End If
            End If
        Next i

    End If

Error:
End Sub

Public Sub IFeatureCallback_OnPLCDisconnect(ByVal ErrorID As Long)
    'set the application as disconnected
    Call Me.ConnectionLost
End Sub

Public Sub ClearPlcData()
    'reset
    m_ConnectionNotifyID = -1
    m_KeyswitchNotifyID = -1
    m_LedNotifyID = -1
    m_PlcNotifyID = -1
    m_ErrorNotifyID = -1

    'release the controller management interface
    Set m_pIfeature = Nothing

    'clear the variants
    m_FeatureNames = Empty
End Sub

Public Function RegisterAllFeatures() As Long
    On Error GoTo Error
    'error
    If m_pIfeature Is Nothing Then
        GoTo Error
    End If

    'init
    Dim strFeatureName As String
    Dim i As Long

    'register to get a notification when the connection to the PLC is lost
    Call m_pIfeature.RegisterForConnectionCheck(Me, m_ConnectionNotifyID,
ErrorID)

    'error?
    If (ErrorID <> 0) Then
        GoTo Error
    End If
    'go thru all features.....
    For i = LBound(m_FeatureNames) To UBound(m_FeatureNames)

        'get the feature name
        strFeatureName = m_FeatureNames(i)

        'if it's one of the features we need...
        If (strFeatureName = FEATURE_KEYSWITCH) Then
            '...register it
            hr = RegisterFeature(strFeatureName, m_KeyswitchNotifyID)
        ElseIf (strFeatureName = FEATURE_LED) Then
            '...register it
            hr = RegisterFeature(strFeatureName, m_LedNotifyID)
        ElseIf (strFeatureName = FEATURE_PLC) Then
            '...register it
            hr = RegisterFeature(strFeatureName, m_PlcNotifyID)
        ElseIf (strFeatureName = FEATURE_ERROR) Then
            '...register it
            hr = RegisterFeature(strFeatureName, m_ErrorNotifyID)
        End If
        'error?
        If (FAILED(hr) = True) Then
            GoTo Error
        End If

    Next i

    RegisterAllFeatures = S_OK
    Exit Function
Error:
    RegisterAllFeatures = E_FAIL
End Function

 Public Sub UnregisterAllFeatures()
    On Error GoTo Error
    'error
    If m_pIfeature Is Nothing Then
        GoTo Error
    End If

    'let's unregister the features => the feature provider stopps sEnding
notIfications
    If (m_KeyswitchNotifyID <> -1) Then
        Call UnregisterFeature(FEATURE_KEYSWITCH, m_KeyswitchNotifyID)
    End If
    If (m_LedNotifyID <> -1) Then
        Call UnregisterFeature(FEATURE_LED, m_LedNotifyID)
    End If
    If (m_PlcNotifyID <> -1) Then
        Call UnregisterFeature(FEATURE_PLC, m_PlcNotifyID)
    End If
    If (m_ErrorNotifyID <> -1) Then
        Call UnregisterFeature(FEATURE_ERROR, m_ErrorNotifyID)
    End If

    'unregister for connection check
    If (m_ConnectionNotifyID <> -1) Then
        Call m_pIfeature.UnregisterForConnectionCheck(m_ConnectionNotifyID,
ErrorID)
    End If
Error:
End Sub

Public Function SetKeyswitch(ByVal Value As String) As Long
    On Error GoTo Error
    'error
    If m_pIfeature Is Nothing Then
        GoTo Error
    End If

    'create value array
    Dim AttributeNames(0 To 0) As String
    Dim AttributeValues(0 To 0) As String
    AttributeNames(0) = ATT_KEYSWITCH
    AttributeValues(0) = Value

    'set the keyswitch
    Call m_pIfeature.SetFeature(FEATURE_KEYSWITCH, AttributeNames,
AttributeValues, ErrorID)

    'error?
    If (ErrorID <> 0) Then
        GoTo Error
    End If

    SetKeyswitch = S_OK
    Exit Function
Error:
    SetKeyswitch = E_FAIL
End Function

Public Function GetCpuType(ByRef Value As String) As Long
    On Error GoTo Error
    'error
    If m_pIfeature Is Nothing Then
        GoTo Error
    End If

    'init
    Value = ""
    Dim AttributeNames As Variant
    Dim AttributeValues As Variant
    Dim i As Long

    'get the personality feature attribute values
    Call m_pIfeature.GetFeature(FEATURE_PERSONALITY, AttributeNames,
AttributeValues, ErrorID)

    'error?
    If (ErrorID <> 0) Then
        GoTo Error
    End If
    'loop thru all attributes
    For i = LBound(AttributeNames) To UBound(AttributeNames)

        'if it's the plc type attribute
        If (AttributeNames(i) = ATT_PERSONALITY_TYPE) Then

            'get type
            Value = AttributeValues(i)
            GetCpuType = S_OK
            Exit Function

        End If

    Next i
    GetCpuType = S_False
    Exit Function
Error:
    GetCpuType = E_FAIL
End Function

Private Function RegisterFeature(ByVal FeatureName As String, ByRef
NotIficationID As Long) As Long
    On Error GoTo Error
    'error
    If m_pIfeature Is Nothing Then
        GoTo Error
    End If

    'init
    Dim var As Variant

    'register the feature to get change notifications
    Call m_pIfeature.RegisterFeatureForChange(Me, FeatureName, var,
NotIficationID, ErrorID)

    'error?
    If (ErrorID <> 0) Then
        GoTo Error
    End If

    RegisterFeature = S_OK
    Exit Function
Error:
    RegisterFeature = E_FAIL
End Function

Private Function UnregisterFeature(ByVal FeatureName As String, ByVal
NotIficationID As Long) As Long
    On Error GoTo Error
    'error
    If m_pIfeature Is Nothing Then
        GoTo Error
    End If

    'unregister the feature to cancel notifications
    Call m_pIfeature.UnregisterFeatureForChange(NotIficationID, ErrorID)

    'error?
    If (ErrorID <> 0) Then
        GoTo Error
    End If

    UnregisterFeature = S_OK
    Exit Function
Error:
    UnregisterFeature = E_FAIL
End Function

Public Function ConnectToPlc(ByVal InstanceName As String, ByRef pIfeature As
Ifeature, ByRef FeatureNames As Variant, ByRef AttributeNames As Variant) As
Long
    'init
    Dim ConnectStrings As Variant
    Dim StartInfos As Variant
    Dim strConnectionString As String
    Dim i As Long

    'get connection strings for all available PLCs
    hr = Browse(ConnectStrings, StartInfos)

    'error?
    If (FAILED(hr) = True) Then
        GoTo Error
    End If

    'go thru all returned connection strings and
    'look for the requested instance name
    For i = LBound(ConnectStrings) To UBound(ConnectStrings)

        'get connection string
        strConnectionString = ConnectStrings(i)

        'init
        Dim pos As Long
        Dim size As Long

        'extract the instance name out of the connection string
        pos = InStr(1, strConnectionString, "\", vbTextCompare)
        size = InStr(pos + 1, strConnectionString, "\", vbTextCompare) - pos -
1
        strConnectionString = Mid(strConnectionString, pos + 1, size)

        'if we have found the correct connection string...
        If (strConnectionString = InstanceName) Then

            'get connection string
            strConnectionString = ConnectStrings(i)

            'init
            Set pIfeature = Nothing
            FeatureNames = Empty
            AttributeNames = Empty
            'connect to the PLC
            Call m_pIPlc.Connect(strConnectionString, pIfeature, FeatureNames,
AttributeNames, ErrorID)
```

```
            'error?
            If (ErrorID <> 0) Then
                GoTo Error
            End If

            ConnectToPlc = S_OK
            Exit Function
        End If
    Next i

Error:
    ConnectToPlc = E_FAIL
End Function

Private Function Browse(ByRef ConnectStrings As Variant, ByRef StartInfos As
Variant) As Long
    On Error GoTo Error

    'clear the arrays
    ConnectStrings = Empty
    StartInfos = Empty

    'init
    Dim ErrorID As Long

    'browse for configured and/or running PLCs
    Call m_pIPlc.Browse(ConnectStrings, StartInfos, ErrorID)

    'error?
    If (ErrorID <> 0) Then
        GoTo Error
    End If

    Browse = S_OK
    Exit Function
Error:
    Browse = E_FAIL
End Function
```

Modules

Definitions.bas

```
'possible connect button text settings
Public Const BTN_TXT_CONNECT As String = "Connect"
Public Const BTN_TXT_DISCONNECT As String = "Disconnect"


'possible function return values
Public Const S_OK As Long = 0
Public Const S_FALSE As Long = 1
Public Const E_FAIL As Long = 2

Public Function FAILED(ByVal Result As Long) As Boolean
    If (Result <> S_OK And Result <> S_FALSE) Then
        FAILED = True
    Else
        FAILED = False
    End If
End Function

Public Function SUCCEEDED(ByVal Result As Long) As Boolean
    If (Result = S_OK Or Result = S_FALSE) Then
        SUCCEEDED = True
    Else
        SUCCEEDED = False
    End If
End Function

Public Function SUCCESS(ByVal Result As Long) As Boolean
    If (Result = S_OK) Then
        SUCCESS = True
    Else
        SUCCESS = False
    End If
End Function
```

Feature.bas

```
'***************************************************************************
'*   Copyright                                                             *
'***************************************************************************

'*--------------------------------------------------------------------------
'*
'*   Author      :   Robin Timmermann
'*   Date        :   03/21/03
'*
'*--------------------------------------------------------------------------
'*
'*   Description :   feature/attribute/value string defines
'*
'*--------------------------------------------------------------------------
'*
'*   Modification history:
'*
'***************************************************************************


'----------------------------------------------------------------------------
'    LED
'----------------------------------------------------------------------------
Public Const FEATURE_LED = "LED"

' attribute power LED
Public Const ATT_LED_POWER = "Power"

' attribute battery fault LED
Public Const ATT_LED_BATF = "BatteryFault"

' attribute internal fault LED
Public Const ATT_LED_INTF = "InternalFault"

' attribute external fault LED
Public Const ATT_LED_EXTF = "ExternalFault"

' attribute bus fault LED
Public Const ATT_LED_BUSFAULTCOUNT = "BusFaultCount"
'VAL_LED_BUSFAULTCOUNT                   %d
Public Const ATT_LED_BUSF = "BusFault_"

' attribute force LED
Public Const ATT_LED_FORCE = "Force"

' attribute run LED
Public Const ATT_LED_RUN = "Run"

' attribute stop LED
Public Const ATT_LED_STOP = "Stop"

' value LED
Public Const VAL_LED_ON = "ON"
Public Const VAL_LED_OFF = "OFF"
Public Const VAL_LED_BLINKING2HZ = "Blinking2HZ"
Public Const VAL_LED_BLINKING05HZ = "Blinking05HZ"

'----------------------------------------------------------------------------
'    KeySwitch
'----------------------------------------------------------------------------
Public Const FEATURE_KEYSWITCH = "KeySwitch"

' attribute KeySwitch
Public Const ATT_KEYSWITCH = "Value"
Public Const VAL_KEYSWITCH_MRES = "MRES"
Public Const VAL_KEYSWITCH_STOP = "STOP"
Public Const VAL_KEYSWITCH_RUN = "RUN"
Public Const VAL_KEYSWITCH_RUNP = "RUNP"

' attribute force coldstart
Public Const ATT_KEYSWITCH_FORCECOLDSTART = "ForceColdstart"
'Public Const VAL_ON = "On"
Public Const VAL_OFF = "Off"

'----------------------------------------------------------------------------
'    PLCInstance
'----------------------------------------------------------------------------
Public Const FEATURE_PLCINSTANCE = "PLCInstance"

' attribute PLCINSTANCE
Public Const ATT_PLCINSTANCE = "Value"
Public Const VAL_PLCINSTANCE_CREATE = "Create"
Public Const VAL_PLCINSTANCE_SHUTDOWN = "Shutdown"

'----------------------------------------------------------------------------
'    PLCPower
'----------------------------------------------------------------------------
Public Const FEATURE_PLCPOWER = "PLCPower"

' attribute PLCPower state
Public Const ATT_PLCPOWER = "Value"
Public Const VAL_ON = "On"
'Public Const VAL_OFF = "Off"

'----------------------------------------------------------------------------
'    PLC
'----------------------------------------------------------------------------
Public Const FEATURE_PLC = "PLC"

' attribute PLC
Public Const ATT_PLC = "Value"
Public Const VAL_PLC_CREATED = "Created"
Public Const VAL_PLC_SHUTDOWNED = "Shutdowned"
Public Const VAL_PLC_NOTAVAILABLE = "NotAvailable"

'----------------------------------------------------------------------------
'    StartAtBoot
'----------------------------------------------------------------------------
Public Const FEATURE_STARTATBOOT = "StartAtBoot"

' attribute Start at boot
Public Const ATT_STARTATBOOT = "Value"
```

```
'Public Const VAL_ON = "On"
'Public Const VAL_OFF = "Off"

'----------------------------------------------------------------------------
'    AutoStart
'----------------------------------------------------------------------------
Public Const FEATURE_AUTOSTART = "AutoStart"

' attribute Autostart
Public Const ATT_AUTOSTART = "Value"
'Public Const VAL_ON = "On"
'Public Const VAL_OFF = "Off"

'----------------------------------------------------------------------------
'    AutoLoad
'----------------------------------------------------------------------------
Public Const FEATURE_AUTOLOAD = "AutoLoad"

' attribute Autoload enabled
Public Const ATT_AUTOLOAD = "Value"
'Public Const VAL_ON = "On"
'Public Const VAL_OFF = "Off"

' attribute Keyswitch state after Autoload
Public Const ATT_AUTOLOAD_KEYSWITCH = "KeySwitch"
Public Const VAL_AUTOLOAD_KS_STOP = "STOP"
Public Const VAL_AUTOLOAD_KS_RUN = "RUN"
Public Const VAL_AUTOLOAD_KS_RUNP = "RUNP"

' attribute Buffer for Autoload
Public Const ATT_AUTOLOAD_BUFFER = "Buffer"
'Public Const VAL_AUTOLOAD_BUFFER             %s

' attribute Buffersize for Autoload
Public Const ATT_AUTOLOAD_BUFFERSIZE = "BufferSize"
'Public Const VAL_AUTOLOAD_BUFFERSIZE         %d

' attribute target filename for Autoload
Public Const ATT_AUTOLOAD_TARGETFILE = "TargetFile"
'Public Const VAL_AUTOLOAD_TARGETFILE         %s

'----------------------------------------------------------------------------
'    Security
'----------------------------------------------------------------------------
Public Const FEATURE_SECURITY = "Security"

' attribute actual Password
Public Const ATT_SECURITY_ACTPASSWORD = "Password"
'Public Const VAL_SECURITY_ACTPASSWORD        %s scrammbled

' attribute new Password
Public Const ATT_SECURITY_NEWPASSWORD = "NewPassword"
'Public Const VAL_SECURITY_NEWPASSWORD        %s scrammbled

' attribute Password check
Public Const ATT_SECURITY_PASSWORDCHECK = "Check"
Public Const VAL_SECURITY_PASSWORDCHECK_PASS = "Passed"
Public Const VAL_SECURITY_PASSWORDCHECK_FAILED = "Failed"

' attribute security level
Public Const ATT_SECURITY_LEVEL = "Level"
Public Const VAL_SECURITY_LEVELPASSWORD = "Password"
Public Const VAL_SECURITY_LEVELPASSWORDDISABLED = "PasswordDisabled"
Public Const VAL_SECURITY_LEVELCONFIRMATION = "Confirmation"
Public Const VAL_SECURITY_LEVELNONE = "None"

' attribute Password Prompt Interval hours
Public Const ATT_SECURITY_INTERVALHOURS = "hInterval"
'Public Const VAL_SECURITY_INTERVALHOURS      %d

' attribute Password Prompt Interval minutes
Public Const ATT_SECURITY_INTERVALMINUTES = "mInterval"
'Public Const VAL_SECURITY_INTERVALMINUTES    %d

'---------------------------------------------------------------------------
'    FMR
'---------------------------------------------------------------------------
Public Const FEATURE_FMR = "FMR"

'---------------------------------------------------------------------------
'    MemoryCard file
'---------------------------------------------------------------------------
Public Const FEATURE_MCF = "MemoryCardFile"

' attribute MemoryCard file Buffer
Public Const ATT_MCF_BUFFER = "Buffer"
'VAL_MCF_BUFFER                          %s

' attribute MemoryCard file Size
Public Const ATT_MCF_SIZE = "Size"
'VAL_MCF_SIZE                            %d

'---------------------------------------------------------------------------
'    Priority
'---------------------------------------------------------------------------
Public Const FEATURE_PRIORITY = "Priority"

' attribute Priority value
Public Const ATT_PRIORITY = "Value"
'Public Const VAL_PRIORITY                    %d

Public Const ATT_PRIORITY_LOWERLIMIT = "LowerLimit"
'Public Const VAL_PRIORITY_LOWERLIMIT         %d

Public Const ATT_PRIORITY_UPPERLIMIT = "UpperLimit"
'Public Const VAL_PRIORITY_UPPERLIMIT         %d

Public Const ATT_NORMAL_PRIORITY = "Normal"
'Public Const VAL_NORMAL_PRIORITY             %d

Public Const ATT_CRITICAL_PRIORITY = "Critical"
'Public Const VAL_CRITICAL_PRIORITY               %d

'---------------------------------------------------------------------------
```

```
'    Minimum Sleep Time
'----------------------------------------------------------------------------
Public Const FEATURE_MINSLEEPTIME = "MinSleepTime"

' attribute Minimum SleepTime value
Public Const ATT_MINSLEEPTIME = "Value"
'Public Const VAL_MINSLEEPTIME                %d

Public Const ATT_MINSLEEPTIME_LOWERLIMIT = "LowerLimit"
'Public Const VAL_MINSLEEPTIME_LOWERLIMIT     %d

Public Const ATT_MINSLEEPTIME_UPPERLIMIT = "UpperLimit"
'Public Const VAL_MINSLEEPTIME_UPPERLIMIT     %d

'----------------------------------------------------------------------------
'    Minimum Cycle Time
'----------------------------------------------------------------------------
Public Const FEATURE_MINCYCLETIME = "MinCycleTime"

' attribute Minimum CycleTime value
Public Const ATT_MINCYCLETIME = "Value"
'Public Const VAL_MINCYCLETIME                %d

Public Const ATT_MINCYCLETIME_LOWERLIMIT = "LowerLimit"
'Public Const VAL_MINSLEEPTIME_LOWERLIMIT     %d

Public Const ATT_MINCYCLETIME_UPPERLIMIT = "UpperLimit"
'Public Const VAL_MINCYCLETIME_UPPERLIMIT     %d

'----------------------------------------------------------------------------
'    CPU Usage
'----------------------------------------------------------------------------
Public Const FEATURE_CPUUSAGE = "Usage"

' attribute Usage count
Public Const ATT_PCUSAGE = "PC"
'VAL_PCUSAGE                             %d [0..100%]

' attribute PLC Usage
Public Const ATT_PLCUSAGE = "PLC"
'Public Const VAL_PLCUSAGE                    %d [0..100%]

' attribute Usage count
Public Const ATT_CPUUSAGE_COUNT = "CPUCount"
'VAL_CPUUSAGE_COUNT                      %d

' attribute CPU Usage values
Public Const ATT_CPUUSAGE = "CPU_"
'Public Const VAL_CPUUSAGE                    %d [0..100%]

'----------------------------------------------------------------------------
'    Timing
'----------------------------------------------------------------------------
Public Const FEATURE_TIMING = "Timing"

' attribute Timing Upper Limit
Public Const ATT_TIMING_UPPERLIMIT = "UpperLimit"
```

```
'VAL_TIMING_UPPERLIMIT                   %d

' attribute CycleTime count
Public Const ATT_TIMING_CYCLETIMECOUNT = "CycleTimeCount"
'Public Const VAL_TIMING_CYCLETIMECOUNT       %d

' attribute CycleTime buffer
Public Const ATT_TIMING_CYCLETIMEBUFFER = "CycleTimeBuffer"
'Public Const VAL_TIMING_CYCLETIMEBUFFER      %d %d...

' attribute CycleTime minimum
Public Const ATT_TIMING_CYCLETIMEMIN = "CycleTimeMin"
'Public Const VAL_TIMING_CYCLETIMEMIN         %d

' attribute CycleTime maximum
Public Const ATT_TIMING_CYCLETIMEMAX = "CycleTimeMax"
'Public Const VAL_TIMING_CYCLETIMEMAX         %d

' attribute CycleTime average
Public Const ATT_TIMING_CYCLETIMEAVE = "CycleTimeAverage"
'Public Const VAL_TIMING_CYCLETIMEAVE         %d

' attribute CycleTime last
Public Const ATT_TIMING_CYCLETIMELAST = "CycleTimeLast"
'Public Const VAL_TIMING_CYCLETIMELAST        %d

' attribute ExecTime minimum
Public Const ATT_TIMING_EXECTIMEMIN = "ExecTimeMin"
'Public Const VAL_TIMING_EXECTIMEMIN          %d

' attribute ExecTime maximum
Public Const ATT_TIMING_EXECTIMEMAX = "ExecTimeMax"
'Public Const VAL_TIMING_EXECTIMEMAX          %d

' attribute ExecTime average
Public Const ATT_TIMING_EXECTIMEAVE = "ExecTimeAverage"
'Public Const VAL_TIMING_EXECTIMEAVE          %d

' attribute ExecTime last
Public Const ATT_TIMING_EXECTIMELAST = "ExecTimeLast"
'Public Const VAL_TIMING_EXECTIMELAST         %d

' attribute SleepIntervalCounter
Public Const ATT_TIMING_SLEEPINTERVALCOUNTER = "SleepIntervalCounter"
'VAL_TIMING_SLEEPINTERVALCOUNTER         %d

' attribute Clear
Public Const ATT_TIMING_CLEAR = "Clear"
'Public Const VAL_TIMING_CLEAR                empty

'----------------------------------------------------------------------------
'    OBExecution
'----------------------------------------------------------------------------
Public Const FEATURE_OBEXEC = "OBExecution"

' attribute WakeInterval
Public Const ATT_OBEXEC_WAKEINTERVAL = "WakeInterval"
```

```
'VAL_OBEXEC_WAKEINTERVAL                  %d

' attribute SleepInterval
Public Const ATT_OBEXEC_SLEEPINTERVAL = "SleepInterval"
'VAL_OBEXEC_SLEEPINTERVAL                 %d

' attribute default WakeInterval
Public Const ATT_OBEXEC_DEFAULTWAKEINTERVAL = "DefaultWakeInterval"
'VAL_OBEXEC_DEFAULTWAKEINTERVAL          %d

' attribute default SleepInterval
Public Const ATT_OBEXEC_DEFAULTSLEEPINTERVAL = "DefaultSleepInterval"
'VAL_OBEXEC_DEFAULTSLEEPINTERVAL         %d

' attribute UpperLimit maximum execution load
Public Const ATT_OBEXEC_UPPERLIMIT = "UpperLimit"
'VAL_OBEXEC_UPPERLIMIT                    %d [0..100%]

' attribute Lowerimit maximum execution load
Public Const ATT_OBEXEC_LOWERLIMIT = "LowerLimit"
'VAL_OBEXEC_LOWERLIMIT                    %d [0..100%]

'----------------------------------------------------------------------------
'    Diagnostic Language
'----------------------------------------------------------------------------
Public Const FEATURE_DIAGLANGUAGE = "DiagnosticLanguage"

' attribute Language count
Public Const ATT_DIAGLANGUAGE_COUNT = "Count"
'VAL_DIAGLANGUAGE_COUNT                   %d

' attribute Language
Public Const ATT_DIAGLANGUAGE = "Language_"
Public Const VAL_LANGUAGE_GERMAN = "GERMAN"
Public Const VAL_LANGUAGE_ENGLISH = "ENGLISH"
Public Const VAL_LANGUAGE_FRENCH = "FRENCH"
Public Const VAL_LANGUAGE_ITALIAN = "ITALIAN"
Public Const VAL_LANGUAGE_SPANISH = "SPANISH"
Public Const VAL_LANGUAGE_JAPANESE = "JAPANESE"
Public Const VAL_LANGUAGE_CHINESE = "CHINESE"

'----------------------------------------------------------------------------
'    Diagnostic Information
'----------------------------------------------------------------------------
Public Const FEATURE_DIAGNOSTIC = "Diagnostic"

' attribute Language
Public Const ATT_DIAG_LANGUAGE = "Language"
'VAL_DIAG_LANGUAGE                        %s

' attribute Diagnostic entry count
Public Const ATT_DIAG_COUNT = "Count"
'VAL_DIAG_COUNT                           %d

' attribute Time
Public Const ATT_DIAG_TIME = "Time_"
'VAL_DIAG_TIME                            %s
```

```
' attribute Date
Public Const ATT_DIAG_DATE = "Date_"
'VAL_DIAG_DATE                           %s

' attribute Event short text
Public Const ATT_DIAG_EVENTSHORT = "EventShort_"
'VAL_DIAG_EVENTSHORT                     %s

' attribute Event long text
Public Const ATT_DIAG_EVENTLONG = "EventLong_"
'VAL_DIAG_EVENTLONG                      %s

' attribute Event ID
Public Const ATT_DIAG_EVENTID = "EventID_"
'VAL_DIAG_EVENTID                        %s

' attribute Event hex text
Public Const ATT_DIAG_EVENTHEX = "EventHex_"
'VAL_DIAG_EVENTHEX                       %s

'---------------------------------------------------------------------------
'    Personality
'---------------------------------------------------------------------------
Public Const FEATURE_PERSONALITY = "Personality"

' attribute instance name
Public Const ATT_PERSONALITY_NAME = "Name"
'VAL_PERSONALITY_NAME                    %s

' attribute CPU type
Public Const ATT_PERSONALITY_TYPE = "Type"
'ATT_PERSONALITY_TYPE                    %s

' attribute product code type
Public Const ATT_PERSONALITY_PRODUCTCODE = "ProductCode"
'ATT_PERSONALITY_PRODUCTCODE             %s

' attribute SW release
Public Const ATT_PERSONALITY_SW_RELEASE = "SWRelease"
'ATT_PERSONALITY_SW_RELEASE              %s

' attribute FW release
Public Const ATT_PERSONALITY_FW_RELEASE = "FWRelease"
'ATT_PERSONALITY_FW_RELEASE              %s

' attribute HW release
Public Const ATT_PERSONALITY_HW_RELEASE = "HWRelease"
'ATT_PERSONALITY_HW_RELEASE              %s

' attribute SLOT Number
Public Const ATT_PERSONALITY_SLOT_NUMBER = "Slot"
'ATT_PERSONALITY_SLOT_NUMBER             %s

' attribute RACK Number
Public Const ATT_PERSONALITY_RACK_NUMBER = "Rack"
'ATT_PERSONALITY_RACK_NUMBER             %s
```

```
' attribute OWNER Info
Public Const ATT_PERSONALITY_OWNER_INFO = "Owner"
'ATT_PERSONALITY_OWNER_INFO              %s

' attribute RACK Number
Public Const ATT_PERSONALITY_COMPANY_INFO = "Company"
'ATT_PERSONALITY_COMPANY_INFO            %s

'---------------------------------------------------------------------------
'    Error
'---------------------------------------------------------------------------
Public Const FEATURE_ERROR = "Error"

' attribute error identification
Public Const ATT_ERROR_ID = "ID"
'VAL_ERROR_ID                            %d

' valid error identifications
Public Enum PSERR
    PSERR_OKAY = 0
    PSERR_NO_MEMORY
    PSERR_ARCHIVE_NOT_VALID_IN_RUN
    PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU
    PSERR_RESTORE_CANNOT_LINKIN_BLOCK
    PSERR_RESTORE_NOT_VALID_IN_RUN
    PSERR_RESTORE_FILE_INVALID
    PSERR_INIT_EDBSERVER
    PSERR_INIT_PDH
    PSERR_KEYSWITCH_NOT_ALLOWED_IN_MCF_OP
    PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU_PASSWORD_PROTECTED
End Enum

'---------------------------------------------------------------------------
'    HelpInfo
'---------------------------------------------------------------------------
Public Const FEATURE_CONTROLLER_HELP = "ControllerHelp"

' attribute host
Public Const ATT_HELP_HOST = "Host"
'VAL_HELP_HOST                           %s

' attribute helpsystem
Public Const ATT_HELP_SYSTEM = "HelpSystem"
Public Const VAL_HELP_SYSTEM_WIN = "WinHelp"
Public Const VAL_HELP_SYSTEM_WEB = "WebHelp"
Public Const VAL_HELP_SYSTEM_HTML = "HTMLHelp"

' attribute document help directory; includes drive letter
Public Const ATT_HELP_DIR = "HelpDir"
'VAL_HELP_DIR                            %s

' attribute Language count
Public Const ATT_HELPLANGUAGE_COUNT = "Count"
'VAL_HELPLANGUAGE_COUNT                  %d

' attribute Language
```

```
Public Const ATT_HELPLANGUAGE = "Language_"
'
'Public Const VAL_LANGUAGE_GERMAN = "GERMAN"
'Public Const VAL_LANGUAGE_ENGLISH = "ENGLISH"
'Public Const VAL_LANGUAGE_FRENCH = "FRENCH"
'Public Const VAL_LANGUAGE_ITALIAN = "ITALIAN"
'Public Const VAL_LANGUAGE_SPANISH = "SPANISH"
'Public Const VAL_LANGUAGE_JAPANESE = "JAPANESE"
'Public Const VAL_LANGUAGE_CHINESE = "CHINESE"


'----------------------------------------------------------------------------
'    CPU Language
'----------------------------------------------------------------------------
Public Const FEATURE_CPULANGUAGE = "CPULanguage"

' attribute Language
Public Const ATT_CPULANGUAGE_CURRENT = "CurrentLanguage"
'
'Public Const VAL_LANGUAGE_GERMAN = "GERMAN"
'Public Const VAL_LANGUAGE_ENGLISH = "ENGLISH"
'Public Const VAL_LANGUAGE_FRENCH = "FRENCH"
'Public Const VAL_LANGUAGE_ITALIAN = "ITALIAN"
'Public Const VAL_LANGUAGE_SPANISH = "SPANISH"
'Public Const VAL_LANGUAGE_JAPANESE = "JAPANESE"
'Public Const VAL_LANGUAGE_CHINESE = "CHINESE"


' attribute Language count
Public Const ATT_CPULANGUAGE_COUNT = "Count"
'VAL_CPULANGUAGE_COUNT                   %d

' attribute Language
Public Const ATT_CPULANGUAGE = "Language_"
'
'Public Const VAL_LANGUAGE_GERMAN = "GERMAN"
'Public Const VAL_LANGUAGE_ENGLISH = "ENGLISH"
'Public Const VAL_LANGUAGE_FRENCH = "FRENCH"
'Public Const VAL_LANGUAGE_ITALIAN = "ITALIAN"
'Public Const VAL_LANGUAGE_SPANISH = "SPANISH"
'Public Const VAL_LANGUAGE_JAPANESE = "JAPANESE"
'Public Const VAL_LANGUAGE_CHINESE = "CHINESE"


'----------------------------------------------------------------------------
'    PC/PG Interface
'----------------------------------------------------------------------------
Public Const FEATURE_PCPGINTERFACE = "PC_PG_Interface"
```

StartModule.bas

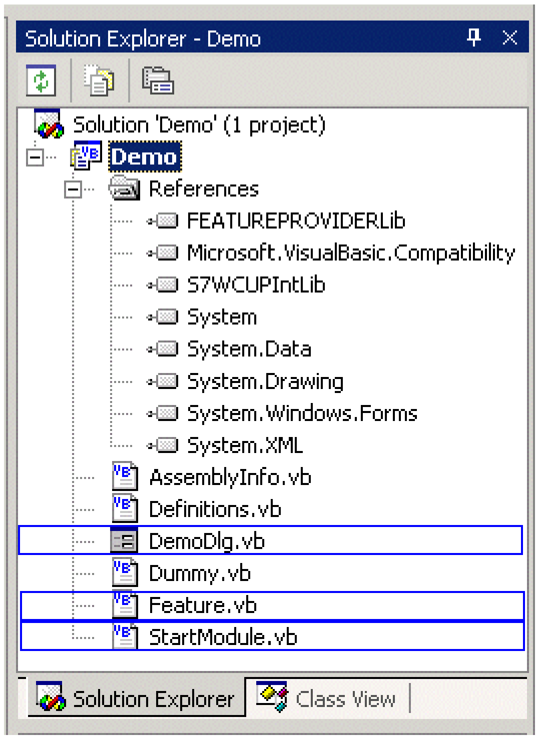
```
Public frmDemoDlg As DemoDlg

Public Sub Main()
    'at this point we create and display the
    'panel dialog
    Set frmDemoDlg = New DemoDlg
    frmDemoDlg.Show
End Sub
```

Visual Basic .NET Sample Program

Introduction to the Visual Basic .NET Demo Panel Program

As described in Introducing the Sample Programs, WinAC ODK installs a Visual Basic .NET implementation of a control panel that can interact with a WinAC controller. This project is located in the ...\Program Files\Siemens\WinAC\ODK\Examples\CMI\CMI_NET_Demo_Panel_VB directory.

You can click one of the indicated files in the VB .NET Solution Explorer picture to access its code listing (online help only):



Demo

DemoDlg.vb

```
Option Strict Off
Option Explicit On
Friend Class DemoDlg
    Inherits System.Windows.Forms.Form
    Implements S7WCUPIntLib.IFeatureCallback

#Region "Windows Form Designer generated code "
    Public Sub New()
        MyBase.New()
        If m_vb6FormDefInstance Is Nothing Then
            If m_InitializingDefInstance Then
                m_vb6FormDefInstance = Me
            Else
                Try
                    'For the start-up form, the first instance created is the
                    default instance.
                    If
                    System.Reflection.Assembly.GetExecutingAssembly.EntryPoint.D
                    eclaringType Is Me.GetType Then
                        m_vb6FormDefInstance = Me
                    End If
                Catch
                End Try
            End If
        End If
        'This call is required by the Windows Form Designer.
        InitializeComponent()
    End Sub
    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal Disposing As Boolean)
        If Disposing Then
            If Not components Is Nothing Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(Disposing)
    End Sub
    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer
    Public ToolTip1 As System.Windows.Forms.ToolTip
    Public WithEvents m_chkRUNPbtn As System.Windows.Forms.CheckBox
    Public WithEvents m_chkRUNbtn As System.Windows.Forms.CheckBox
    Public WithEvents m_chkSTOPbtn As System.Windows.Forms.CheckBox
    Public WithEvents m_chkSTOP As System.Windows.Forms.CheckBox
    Public WithEvents m_chkRUN As System.Windows.Forms.CheckBox
    Public WithEvents m_chkFRCE As System.Windows.Forms.CheckBox
    Public WithEvents m_chkBUSF2 As System.Windows.Forms.CheckBox
    Public WithEvents m_chkBUSF1 As System.Windows.Forms.CheckBox
    Public WithEvents m_chkEXTF As System.Windows.Forms.CheckBox
    Public WithEvents m_chkINTF As System.Windows.Forms.CheckBox
    Public WithEvents m_chkBATF As System.Windows.Forms.CheckBox
    Public WithEvents m_chkON As System.Windows.Forms.CheckBox
    Public WithEvents m_btnMRES As System.Windows.Forms.Button
    Public WithEvents m_btnSTOP As System.Windows.Forms.Button
    Public WithEvents m_btnRUN As System.Windows.Forms.Button
    Public WithEvents m_btnRUNP As System.Windows.Forms.Button
    Public WithEvents m_btnType As System.Windows.Forms.Button
    Public WithEvents m_btnConnect As System.Windows.Forms.Button
    Public WithEvents m_edtType As System.Windows.Forms.TextBox
    Public WithEvents m_edtInstanceName As System.Windows.Forms.TextBox
    Public WithEvents Label5 As System.Windows.Forms.Label
    Public WithEvents Label4 As System.Windows.Forms.Label
    Public WithEvents Label3 As System.Windows.Forms.Label
    Public WithEvents Label2 As System.Windows.Forms.Label
    Public WithEvents Label1 As System.Windows.Forms.Label
    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
    InitializeComponent()
        Dim resources As System.Resources.ResourceManager = New
        System.Resources.ResourceManager(GetType(DemoDlg))
        Me.components = New System.ComponentModel.Container()
        Me.ToolTip1 = New System.Windows.Forms.ToolTip(components)
        Me.ToolTip1.Active = True
        Me.m_chkRUNPbtn = New System.Windows.Forms.CheckBox
        Me.m_chkRUNbtn = New System.Windows.Forms.CheckBox
        Me.m_chkSTOPbtn = New System.Windows.Forms.CheckBox
        Me.m_chkSTOP = New System.Windows.Forms.CheckBox
        Me.m_chkRUN = New System.Windows.Forms.CheckBox
        Me.m_chkFRCE = New System.Windows.Forms.CheckBox
        Me.m_chkBUSF2 = New System.Windows.Forms.CheckBox
        Me.m_chkBUSF1 = New System.Windows.Forms.CheckBox
        Me.m_chkEXTF = New System.Windows.Forms.CheckBox
        Me.m_chkINTF = New System.Windows.Forms.CheckBox
        Me.m_chkBATF = New System.Windows.Forms.CheckBox
        Me.m_chkON = New System.Windows.Forms.CheckBox
        Me.m_btnMRES = New System.Windows.Forms.Button
        Me.m_btnSTOP = New System.Windows.Forms.Button
        Me.m_btnRUN = New System.Windows.Forms.Button
        Me.m_btnRUNP = New System.Windows.Forms.Button
        Me.m_btnType = New System.Windows.Forms.Button
        Me.m_btnConnect = New System.Windows.Forms.Button
        Me.m_edtType = New System.Windows.Forms.TextBox
        Me.m_edtInstanceName = New System.Windows.Forms.TextBox
        Me.Label5 = New System.Windows.Forms.Label
        Me.Label4 = New System.Windows.Forms.Label
        Me.Label3 = New System.Windows.Forms.Label
        Me.Label2 = New System.Windows.Forms.Label
        Me.Label1 = New System.Windows.Forms.Label
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog
        Me.Text = "Demo"
        Me.ClientSize = New System.Drawing.Size(184, 378)
        Me.Location = New System.Drawing.Point(3, 29)
        Me.MaximizeBox = False
        Me.MinimizeBox = False
        Me.ShowInTaskbar = False
        Me.StartPosition =
        System.Windows.Forms.FormStartPosition.WindowsDefaultLocation
        Me.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.BackColor = System.Drawing.SystemColors.Control
        Me.ControlBox = True
        Me.Enabled = True
        Me.KeyPreview = False
        Me.Cursor = System.Windows.Forms.Cursors.Default
        Me.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.HelpButton = False
        Me.WindowState = System.Windows.Forms.FormWindowState.Normal
        Me.Name = "DemoDlg"
        Me.m_chkRUNPbtn.Enabled = False
        Me.m_chkRUNPbtn.Size = New System.Drawing.Size(17, 17)
        Me.m_chkRUNPbtn.Location = New System.Drawing.Point(150, 210)
        Me.m_chkRUNPbtn.TabIndex = 24
        Me.m_chkRUNPbtn.TabStop = False
        Me.m_chkRUNPbtn.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkRUNPbtn.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkRUNPbtn.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkRUNPbtn.Text = ""
        Me.m_chkRUNPbtn.CausesValidation = True
        Me.m_chkRUNPbtn.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkRUNPbtn.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkRUNPbtn.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkRUNPbtn.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkRUNPbtn.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkRUNPbtn.Visible = True
        Me.m_chkRUNPbtn.Name = "m_chkRUNPbtn"
        Me.m_chkRUNbtn.Enabled = False
        Me.m_chkRUNbtn.Size = New System.Drawing.Size(17, 17)
        Me.m_chkRUNbtn.Location = New System.Drawing.Point(150, 231)
        Me.m_chkRUNbtn.TabIndex = 23
        Me.m_chkRUNbtn.TabStop = False
        Me.m_chkRUNbtn.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkRUNbtn.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkRUNbtn.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkRUNbtn.Text = ""
        Me.m_chkRUNbtn.CausesValidation = True
        Me.m_chkRUNbtn.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkRUNbtn.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkRUNbtn.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkRUNbtn.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkRUNbtn.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkRUNbtn.Visible = True
        Me.m_chkRUNbtn.Name = "m_chkRUNbtn"
        Me.m_chkSTOPbtn.Enabled = False
        Me.m_chkSTOPbtn.Size = New System.Drawing.Size(17, 17)
        Me.m_chkSTOPbtn.Location = New System.Drawing.Point(150, 251)
        Me.m_chkSTOPbtn.TabIndex = 22
        Me.m_chkSTOPbtn.TabStop = False
        Me.m_chkSTOPbtn.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkSTOPbtn.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkSTOPbtn.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkSTOPbtn.Text = ""
        Me.m_chkSTOPbtn.CausesValidation = True
        Me.m_chkSTOPbtn.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkSTOPbtn.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkSTOPbtn.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkSTOPbtn.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkSTOPbtn.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkSTOPbtn.Visible = True
        Me.m_chkSTOPbtn.Name = "m_chkSTOPbtn"
        Me.m_chkSTOP.Text = "  STOP"
        Me.m_chkSTOP.Enabled = False
        Me.m_chkSTOP.Size = New System.Drawing.Size(65, 17)
        Me.m_chkSTOP.Location = New System.Drawing.Point(24, 333)
        Me.m_chkSTOP.TabIndex = 21
        Me.m_chkSTOP.TabStop = False
        Me.m_chkSTOP.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkSTOP.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkSTOP.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkSTOP.CausesValidation = True
        Me.m_chkSTOP.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkSTOP.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkSTOP.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkSTOP.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkSTOP.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkSTOP.Visible = True
        Me.m_chkSTOP.Name = "m_chkSTOP"
        Me.m_chkRUN.Text = "  RUN"
        Me.m_chkRUN.Enabled = False
        Me.m_chkRUN.Size = New System.Drawing.Size(65, 17)
        Me.m_chkRUN.Location = New System.Drawing.Point(24, 314)
        Me.m_chkRUN.TabIndex = 20
        Me.m_chkRUN.TabStop = False
        Me.m_chkRUN.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkRUN.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkRUN.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkRUN.CausesValidation = True
        Me.m_chkRUN.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkRUN.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkRUN.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkRUN.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkRUN.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkRUN.Visible = True
        Me.m_chkRUN.Name = "m_chkRUN"
        Me.m_chkFRCE.Text = "  FRCE"
        Me.m_chkFRCE.Enabled = False
        Me.m_chkFRCE.Size = New System.Drawing.Size(65, 17)
        Me.m_chkFRCE.Location = New System.Drawing.Point(24, 294)
        Me.m_chkFRCE.TabIndex = 19
        Me.m_chkFRCE.TabStop = False
        Me.m_chkFRCE.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkFRCE.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkFRCE.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkFRCE.CausesValidation = True
        Me.m_chkFRCE.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkFRCE.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkFRCE.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkFRCE.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkFRCE.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkFRCE.Visible = True
        Me.m_chkFRCE.Name = "m_chkFRCE"
        Me.m_chkBUSF2.Text = "  BUSF2"
        Me.m_chkBUSF2.Enabled = False
        Me.m_chkBUSF2.Size = New System.Drawing.Size(65, 17)
        Me.m_chkBUSF2.Location = New System.Drawing.Point(24, 275)
        Me.m_chkBUSF2.TabIndex = 18
        Me.m_chkBUSF2.TabStop = False
        Me.m_chkBUSF2.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkBUSF2.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkBUSF2.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkBUSF2.CausesValidation = True
        Me.m_chkBUSF2.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkBUSF2.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkBUSF2.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkBUSF2.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkBUSF2.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkBUSF2.Visible = True
        Me.m_chkBUSF2.Name = "m_chkBUSF2"
        Me.m_chkBUSF1.Text = "  BUSF1"
        Me.m_chkBUSF1.Enabled = False
        Me.m_chkBUSF1.Size = New System.Drawing.Size(65, 17)
        Me.m_chkBUSF1.Location = New System.Drawing.Point(24, 256)
        Me.m_chkBUSF1.TabIndex = 17
        Me.m_chkBUSF1.TabStop = False
        Me.m_chkBUSF1.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkBUSF1.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkBUSF1.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkBUSF1.CausesValidation = True
        Me.m_chkBUSF1.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkBUSF1.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkBUSF1.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkBUSF1.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkBUSF1.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkBUSF1.Visible = True
        Me.m_chkBUSF1.Name = "m_chkBUSF1"
        Me.m_chkEXTF.Text = "  EXTF"
        Me.m_chkEXTF.Enabled = False
        Me.m_chkEXTF.Size = New System.Drawing.Size(57, 17)
        Me.m_chkEXTF.Location = New System.Drawing.Point(24, 236)
        Me.m_chkEXTF.TabIndex = 16
        Me.m_chkEXTF.TabStop = False
        Me.m_chkEXTF.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkEXTF.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkEXTF.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkEXTF.CausesValidation = True
        Me.m_chkEXTF.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkEXTF.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkEXTF.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkEXTF.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkEXTF.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkEXTF.Visible = True
        Me.m_chkEXTF.Name = "m_chkEXTF"
        Me.m_chkINTF.Text = "  INTF"
        Me.m_chkINTF.Enabled = False
        Me.m_chkINTF.Size = New System.Drawing.Size(57, 17)
        Me.m_chkINTF.Location = New System.Drawing.Point(24, 217)
        Me.m_chkINTF.TabIndex = 15
        Me.m_chkINTF.TabStop = False
        Me.m_chkINTF.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkINTF.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkINTF.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkINTF.CausesValidation = True
        Me.m_chkINTF.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkINTF.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkINTF.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkINTF.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkINTF.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkINTF.Visible = True
        Me.m_chkINTF.Name = "m_chkINTF"
        Me.m_chkBATF.Text = "  BATF"
        Me.m_chkBATF.Enabled = False
        Me.m_chkBATF.Size = New System.Drawing.Size(73, 17)
        Me.m_chkBATF.Location = New System.Drawing.Point(24, 179)
        Me.m_chkBATF.TabIndex = 14
        Me.m_chkBATF.TabStop = False
        Me.m_chkBATF.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkBATF.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkBATF.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkBATF.CausesValidation = True
        Me.m_chkBATF.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkBATF.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkBATF.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkBATF.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkBATF.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkBATF.Visible = True
        Me.m_chkBATF.Name = "m_chkBATF"
        Me.m_chkON.Text = "  ON"
        Me.m_chkON.Enabled = False
        Me.m_chkON.Size = New System.Drawing.Size(65, 17)
        Me.m_chkON.Location = New System.Drawing.Point(24, 157)
        Me.m_chkON.TabIndex = 13
        Me.m_chkON.TabStop = False
        Me.m_chkON.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_chkON.CheckAlign = System.Drawing.ContentAlignment.MiddleLeft
        Me.m_chkON.BackColor = System.Drawing.SystemColors.Control
        Me.m_chkON.CausesValidation = True
        Me.m_chkON.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_chkON.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_chkON.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_chkON.Appearance = System.Windows.Forms.Appearance.Normal
        Me.m_chkON.CheckState = System.Windows.Forms.CheckState.Unchecked
        Me.m_chkON.Visible = True
        Me.m_chkON.Name = "m_chkON"
        Me.m_btnMRES.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        Me.m_btnMRES.Text = "MRES"
        Me.m_btnMRES.Size = New System.Drawing.Size(44, 20)
        Me.m_btnMRES.Location = New System.Drawing.Point(102, 328)
        Me.m_btnMRES.TabIndex = 7
        Me.m_btnMRES.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_btnMRES.BackColor = System.Drawing.SystemColors.Control
        Me.m_btnMRES.CausesValidation = True
        Me.m_btnMRES.Enabled = True
        Me.m_btnMRES.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_btnMRES.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_btnMRES.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_btnMRES.TabStop = True
        Me.m_btnMRES.Name = "m_btnMRES"
        Me.m_btnSTOP.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        Me.m_btnSTOP.Text = "STOP"
        Me.m_btnSTOP.Size = New System.Drawing.Size(44, 20)
        Me.m_btnSTOP.Location = New System.Drawing.Point(102, 249)
        Me.m_btnSTOP.TabIndex = 6
        Me.m_btnSTOP.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_btnSTOP.BackColor = System.Drawing.SystemColors.Control
        Me.m_btnSTOP.CausesValidation = True
        Me.m_btnSTOP.Enabled = True
        Me.m_btnSTOP.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_btnSTOP.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_btnSTOP.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_btnSTOP.TabStop = True
        Me.m_btnSTOP.Name = "m_btnSTOP"
        Me.m_btnRUN.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        Me.m_btnRUN.Text = "RUN"
        Me.m_btnRUN.Size = New System.Drawing.Size(44, 20)
        Me.m_btnRUN.Location = New System.Drawing.Point(102, 229)
        Me.m_btnRUN.TabIndex = 5
        Me.m_btnRUN.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_btnRUN.BackColor = System.Drawing.SystemColors.Control
        Me.m_btnRUN.CausesValidation = True
        Me.m_btnRUN.Enabled = True
        Me.m_btnRUN.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_btnRUN.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_btnRUN.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_btnRUN.TabStop = True
        Me.m_btnRUN.Name = "m_btnRUN"
        Me.m_btnRUNP.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        Me.m_btnRUNP.Text = "RUN-P"
        Me.m_btnRUNP.Size = New System.Drawing.Size(44, 20)
        Me.m_btnRUNP.Location = New System.Drawing.Point(102, 208)
        Me.m_btnRUNP.TabIndex = 4
        Me.m_btnRUNP.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_btnRUNP.BackColor = System.Drawing.SystemColors.Control
        Me.m_btnRUNP.CausesValidation = True
        Me.m_btnRUNP.Enabled = True
        Me.m_btnRUNP.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_btnRUNP.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_btnRUNP.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_btnRUNP.TabStop = True
        Me.m_btnRUNP.Name = "m_btnRUNP"
        Me.m_btnType.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
        Me.m_btnType.Text = "Get Type"
        Me.m_btnType.Size = New System.Drawing.Size(72, 22)
        Me.m_btnType.Location = New System.Drawing.Point(102, 72)
        Me.m_btnType.TabIndex = 3
        Me.m_btnType.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_btnType.BackColor = System.Drawing.SystemColors.Control
        Me.m_btnType.CausesValidation = True
        Me.m_btnType.Enabled = True
        Me.m_btnType.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_btnType.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_btnType.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_btnType.TabStop = True
        Me.m_btnType.Name = "m_btnType"
        Me.m_btnConnect.TextAlign =
        System.Drawing.ContentAlignment.MiddleCenter
        Me.m_btnConnect.Text = "Connect"
        Me.m_btnConnect.Size = New System.Drawing.Size(72, 22)
        Me.m_btnConnect.Location = New System.Drawing.Point(102, 30)
        Me.m_btnConnect.TabIndex = 2
        Me.m_btnConnect.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_btnConnect.BackColor = System.Drawing.SystemColors.Control
        Me.m_btnConnect.CausesValidation = True
        Me.m_btnConnect.Enabled = True
        Me.m_btnConnect.ForeColor = System.Drawing.SystemColors.ControlText
        Me.m_btnConnect.Cursor = System.Windows.Forms.Cursors.Default
        Me.m_btnConnect.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_btnConnect.TabStop = True
        Me.m_btnConnect.Name = "m_btnConnect"
        Me.m_edtType.AutoSize = False
        Me.m_edtType.Enabled = False
        Me.m_edtType.Size = New System.Drawing.Size(81, 22)
        Me.m_edtType.Location = New System.Drawing.Point(14, 72)
        Me.m_edtType.ReadOnly = True
        Me.m_edtType.TabIndex = 9
        Me.m_edtType.TabStop = False
        Me.m_edtType.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_edtType.AcceptsReturn = True
        Me.m_edtType.TextAlign = System.Windows.Forms.HorizontalAlignment.Left
        Me.m_edtType.BackColor = System.Drawing.SystemColors.Window
        Me.m_edtType.CausesValidation = True
        Me.m_edtType.ForeColor = System.Drawing.SystemColors.WindowText
        Me.m_edtType.HideSelection = True
        Me.m_edtType.Maxlength = 0
        Me.m_edtType.Cursor = System.Windows.Forms.Cursors.IBeam
        Me.m_edtType.MultiLine = False
        Me.m_edtType.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_edtType.ScrollBars = System.Windows.Forms.ScrollBars.None
        Me.m_edtType.Visible = True
        Me.m_edtType.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.m_edtType.Name = "m_edtType"
        Me.m_edtInstanceName.AutoSize = False
        Me.m_edtInstanceName.Size = New System.Drawing.Size(81, 22)
        Me.m_edtInstanceName.Location = New System.Drawing.Point(14, 30)
        Me.m_edtInstanceName.TabIndex = 1
        Me.m_edtInstanceName.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.m_edtInstanceName.AcceptsReturn = True
        Me.m_edtInstanceName.TextAlign =
        System.Windows.Forms.HorizontalAlignment.Left
        Me.m_edtInstanceName.BackColor = System.Drawing.SystemColors.Window
        Me.m_edtInstanceName.CausesValidation = True
        Me.m_edtInstanceName.Enabled = True
        Me.m_edtInstanceName.ForeColor = System.Drawing.SystemColors.WindowText
        Me.m_edtInstanceName.HideSelection = True
        Me.m_edtInstanceName.ReadOnly = False
        Me.m_edtInstanceName.Maxlength = 0
        Me.m_edtInstanceName.Cursor = System.Windows.Forms.Cursors.IBeam
        Me.m_edtInstanceName.MultiLine = False
        Me.m_edtInstanceName.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.m_edtInstanceName.ScrollBars = System.Windows.Forms.ScrollBars.None
        Me.m_edtInstanceName.TabStop = True
        Me.m_edtInstanceName.Visible = True
        Me.m_edtInstanceName.BorderStyle =
        System.Windows.Forms.BorderStyle.Fixed3D
        Me.m_edtInstanceName.Name = "m_edtInstanceName"
        Me.Label5.Text = "CPU"
        Me.Label5.Size = New System.Drawing.Size(41, 17)
        Me.Label5.Location = New System.Drawing.Point(14, 202)
        Me.Label5.TabIndex = 12
        Me.Label5.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.Label5.TextAlign = System.Drawing.ContentAlignment.TopLeft
        Me.Label5.BackColor = System.Drawing.SystemColors.Control
        Me.Label5.Enabled = True
        Me.Label5.ForeColor = System.Drawing.SystemColors.ControlText
        Me.Label5.Cursor = System.Windows.Forms.Cursors.Default
        Me.Label5.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.Label5.UseMnemonic = True
        Me.Label5.Visible = True
        Me.Label5.AutoSize = False
        Me.Label5.BorderStyle = System.Windows.Forms.BorderStyle.None
        Me.Label5.Name = "Label5"
        Me.Label4.Text = "PS"
        Me.Label4.Size = New System.Drawing.Size(65, 17)
        Me.Label4.Location = New System.Drawing.Point(14, 138)
        Me.Label4.TabIndex = 11
        Me.Label4.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.Label4.TextAlign = System.Drawing.ContentAlignment.TopLeft
        Me.Label4.BackColor = System.Drawing.SystemColors.Control
        Me.Label4.Enabled = True
        Me.Label4.ForeColor = System.Drawing.SystemColors.ControlText
        Me.Label4.Cursor = System.Windows.Forms.Cursors.Default
        Me.Label4.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.Label4.UseMnemonic = True
        Me.Label4.Visible = True
        Me.Label4.AutoSize = False
        Me.Label4.BorderStyle = System.Windows.Forms.BorderStyle.None
        Me.Label4.Name = "Label4"
        Me.Label3.Text = "SIEMENS"
        Me.Label3.Size = New System.Drawing.Size(105, 17)
        Me.Label3.Location = New System.Drawing.Point(14, 120)
        Me.Label3.TabIndex = 10
        Me.Label3.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.Label3.TextAlign = System.Drawing.ContentAlignment.TopLeft
        Me.Label3.BackColor = System.Drawing.SystemColors.Control
        Me.Label3.Enabled = True
        Me.Label3.ForeColor = System.Drawing.SystemColors.ControlText
        Me.Label3.Cursor = System.Windows.Forms.Cursors.Default
        Me.Label3.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.Label3.UseMnemonic = True
        Me.Label3.Visible = True
        Me.Label3.AutoSize = False
        Me.Label3.BorderStyle = System.Windows.Forms.BorderStyle.None
        Me.Label3.Name = "Label3"
        Me.Label2.Text = "PLC type:"
        Me.Label2.Size = New System.Drawing.Size(73, 17)
        Me.Label2.Location = New System.Drawing.Point(14, 55)
        Me.Label2.TabIndex = 8
        Me.Label2.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.Label2.TextAlign = System.Drawing.ContentAlignment.TopLeft
        Me.Label2.BackColor = System.Drawing.SystemColors.Control
        Me.Label2.Enabled = True
        Me.Label2.ForeColor = System.Drawing.SystemColors.ControlText
        Me.Label2.Cursor = System.Windows.Forms.Cursors.Default
        Me.Label2.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.Label2.UseMnemonic = True
        Me.Label2.Visible = True
```

```
        Me.Label2.AutoSize = False
        Me.Label2.BorderStyle = System.Windows.Forms.BorderStyle.None
        Me.Label2.Name = "Label2"
        Me.Label1.Text = "Instance name:"
        Me.Label1.Size = New System.Drawing.Size(89, 17)
        Me.Label1.Location = New System.Drawing.Point(14, 10)
        Me.Label1.TabIndex = 0
        Me.Label1.Font = New System.Drawing.Font("Arial", 8!,
        System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
        CType(0, Byte))
        Me.Label1.TextAlign = System.Drawing.ContentAlignment.TopLeft
        Me.Label1.BackColor = System.Drawing.SystemColors.Control
        Me.Label1.Enabled = True
        Me.Label1.ForeColor = System.Drawing.SystemColors.ControlText
        Me.Label1.Cursor = System.Windows.Forms.Cursors.Default
        Me.Label1.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.Label1.UseMnemonic = True
        Me.Label1.Visible = True
        Me.Label1.AutoSize = False
        Me.Label1.BorderStyle = System.Windows.Forms.BorderStyle.None
        Me.Label1.Name = "Label1"
        Me.Controls.Add(m_chkRUNPbtn)
        Me.Controls.Add(m_chkRUNbtn)
        Me.Controls.Add(m_chkSTOPbtn)
        Me.Controls.Add(m_chkSTOP)
        Me.Controls.Add(m_chkRUN)
        Me.Controls.Add(m_chkFRCE)
        Me.Controls.Add(m_chkBUSF2)
        Me.Controls.Add(m_chkBUSF1)
        Me.Controls.Add(m_chkEXTF)
        Me.Controls.Add(m_chkINTF)
        Me.Controls.Add(m_chkBATF)
        Me.Controls.Add(m_chkON)
        Me.Controls.Add(m_btnMRES)
        Me.Controls.Add(m_btnSTOP)
        Me.Controls.Add(m_btnRUN)
        Me.Controls.Add(m_btnRUNP)
        Me.Controls.Add(m_btnType)
        Me.Controls.Add(m_btnConnect)
        Me.Controls.Add(m_edtType)
        Me.Controls.Add(m_edtInstanceName)
        Me.Controls.Add(Label5)
        Me.Controls.Add(Label4)
        Me.Controls.Add(Label3)
        Me.Controls.Add(Label2)
        Me.Controls.Add(Label1)
    End Sub
#End Region
#Region "Upgrade Support "
    Private Shared m_vb6FormDefInstance As DemoDlg
    Private Shared m_InitializingDefInstance As Boolean
    Public Shared Property DefInstance() As DemoDlg
        Get
            If m_vb6FormDefInstance Is Nothing OrElse
            m_vb6FormDefInstance.IsDisposed Then
                m_InitializingDefInstance = True
                m_vb6FormDefInstance = New DemoDlg()
                m_InitializingDefInstance = False
            End If
            DefInstance = m_vb6FormDefInstance
        End Get
        Set
            m_vb6FormDefInstance = Value
        End Set
    End Property
#End Region

    Private hr As Integer
    Private ErrorID As Integer
    Private m_pIPlc As S7WCUPIntLib.IPLC
    Private m_pIFeature As S7WCUPIntLib.IFeature
    Private m_FeatureNames As Object
    Private m_ConnectionNotifyID As Integer
    Private m_KeyswitchNotifyID As Integer
    Private m_LedNotifyID As Integer
    Private m_PlcNotifyID As Integer
    Private m_ErrorNotifyID As Integer

    Private Sub ActivateControls()
        'activate buttons
        m_btnRUNP.Enabled = True
        m_btnRUN.Enabled = True
        m_btnSTOP.Enabled = True
        m_btnMRES.Enabled = True
        m_btnType.Enabled = True
    End Sub

    Private Sub DeactivateControls()
        'uncheck all radio buttons
        m_chkON.CheckState = False
        m_chkBATF.CheckState = False
        m_chkINTF.CheckState = False
        m_chkEXTF.CheckState = False
        m_chkBUSF1.CheckState = False
        m_chkBUSF2.CheckState = False
        m_chkFRCE.CheckState = False
        m_chkRUN.CheckState = False
        m_chkSTOP.CheckState = False
        m_chkRUNPbtn.CheckState = False
        m_chkRUNbtn.CheckState = False
        m_chkSTOPbtn.CheckState = False

        'deactivate buttons
        m_btnRUNP.Enabled = False
        m_btnRUN.Enabled = False
        m_btnSTOP.Enabled = False
        m_btnMRES.Enabled = False
        m_btnType.Enabled = False
    End Sub

    Private Function Connect() As Integer
        'init
        Dim AttributeNames As Object
        'get the instance name
        Dim strInstanceName As String
        strInstanceName = m_edtInstanceName.Text
        'disable the instance name box
        m_edtInstanceName.Enabled = False
        'release all PLC related data
        Call Me.ClearPlcData()
        'connect to the PLC
        hr = Me.ConnectToPlc(strInstanceName, m_pIFeature, m_FeatureNames,
        AttributeNames)

        'if connection succeeded...
        If (SUCCEEDED(hr) = True) Then

            'register all necessary features
            hr = Me.RegisterAllFeatures()

        End If

        Connect = hr
    End Function

    Private Sub Disconnect()
        'unregister all features
        Call Me.UnregisterAllFeatures()

        'clear all PLC related data
        Call Me.ClearPlcData()

        'deactivate all controls
        Call DeactivateControls()

        'clear the plc type info
        m_edtType.Text = ""
        'enable the instance name box
        m_edtInstanceName.Enabled = True
    End Sub

    Public Sub ConnectionLost()
        'clear the plc proxy
        Call Me.ClearPlcData()

        'deactivate all controls
        Call DeactivateControls()

        'change the connect button
        m_btnConnect.Text = BTN_TXT_CONNECT

        'disable the connect button => app needs to be restartet
        m_btnConnect.Enabled = False

        'clear the plc type info
        m_edtType.Text = ""
    End Sub

    Private Sub m_btnConnect_Click(ByVal eventSender As System.Object, ByVal
    eventArgs As System.EventArgs) Handles m_btnConnect.Click
```

```
        'lock the button
        m_btnConnect.Enabled = False

        'get the button text
        Dim strBtnText As String
        strBtnText = m_btnConnect.Text

        'if we want to connect...
        If (strBtnText = BTN_TXT_CONNECT) Then

            'connect to the PLC
            hr = Connect()

            'if everything is fine...
            If (SUCCEEDED(hr) = True) Then
                'change the connect button
                m_btnConnect.Text = BTN_TXT_DISCONNECT

            'if connection failed...
            Else
                'disconnect the PLC and clear all settings
                Call Disconnect()

                'show error message box
                MsgBox ("Connecting to the PLC failed.")

            End If
        'if we want to disconnect...
        ElseIf (strBtnText = BTN_TXT_DISCONNECT) Then

            'disconnect the PLC
            Call Disconnect()

            'change the connect button
            m_btnConnect.Text = BTN_TXT_CONNECT

        End If

        'unlock the button
        m_btnConnect.Enabled = True
    End Sub

    Private Sub m_btnMRES_Click(ByVal eventSender As System.Object, ByVal
    eventArgs As System.EventArgs) Handles m_btnMRES.Click
        'create message
        Dim Message As String
        Message = "The module will be reset (clear/reset).  All user data will
        be deleted and all existing connections to the module will be
        disconnected.  Do you really want to reset the module?"

        'double-check with the user
        If MsgBox(Message, MsgBoxStyle.YesNo + MsgBoxStyle.Exclamation +
        MsgBoxStyle.ApplicationModal, "Demo") = MsgBoxResult.Yes Then
            'send a request to change the keyswitch to MRES
            Me.SetKeyswitch (VAL_KEYSWITCH_MRES)

        End If
        'send a request to change the keyswitch to MRES
        Me.SetKeyswitch (VAL_KEYSWITCH_MRES)
    End Sub

    Private Sub m_btnRUN_Click(ByVal eventSender As System.Object, ByVal
    eventArgs As System.EventArgs) Handles m_btnRUN.Click
        'send a request to change the keyswitch to RUN
        Me.SetKeyswitch (VAL_KEYSWITCH_RUN)
    End Sub

    Private Sub m_btnRUNP_Click(ByVal eventSender As System.Object, ByVal
    eventArgs As System.EventArgs) Handles m_btnRUNP.Click
        'send a request to change the keyswitch to RUN-P
        Me.SetKeyswitch (VAL_KEYSWITCH_RUNP)
    End Sub

    Private Sub m_btnSTOP_Click(ByVal eventSender As System.Object, ByVal
    eventArgs As System.EventArgs) Handles m_btnSTOP.Click
        'send a request to change the keyswitch to STOP
        Me.SetKeyswitch (VAL_KEYSWITCH_STOP)
    End Sub

    Private Sub m_btnType_Click(ByVal eventSender As System.Object, ByVal
    eventArgs As System.EventArgs) Handles m_btnType.Click
        'init
        Dim strPlcType As String

        'get the plc type info
        hr = Me.GetCpuType(strPlcType)

        'error
        If (FAILED(hr) = True) Then
            strPlcType = "error"
        ElseIf (hr = S_FALSE) Then
            strPlcType = "unknown"
        End If

        'display the plc type
        m_edtType.Text = strPlcType
    End Sub

    Private Sub DemoDlg_Load(ByVal eventSender As System.Object, ByVal
    eventArgs As System.EventArgs) Handles MyBase.Load
        'create an instance of the feature provider
        m_pIPlc = New FEATUREPROVIDERLib.PLC

        'set default instance name
        m_edtInstanceName.Text = "WinLC"

        'clear plc type info
        m_edtType.Text = ""

        'set the initial connect button text
        m_btnConnect.Text = BTN_TXT_CONNECT

        'disable all panel controls
```

```
        Call DeactivateControls()
    End Sub

    Private Sub DemoDlg_Closed(ByVal eventSender As System.Object, ByVal
    eventArgs As System.EventArgs) Handles MyBase.Closed
        'disconnect
        Call Disconnect()

        'Release the feature provider
        m_pIPlc = Nothing

    End Sub

    Public Sub IFeatureCallback_OnFeatureChanged(ByVal FeatureName As String,
    ByVal Context As Object, ByVal NotificationID As Integer, ByVal
    AttributeNames As Object, ByVal AttributeValues As Object) Implements
    S7WCUPIntLib.IFeatureCallback.OnFeatureChanged
        'init
        Dim strName As String
        Dim strValue As String
        Dim i As Integer

        'if it's the keyswitch feature...
         Dim strMessage As String
        If (FeatureName = FEATURE_KEYSWITCH) Then
            'go thru all attributes...
            For i = LBound(AttributeNames) To UBound(AttributeNames)

                'get the attribute name
                strName = AttributeNames(i)

                'if it's the keyswitch attribute...
                If (strName = ATT_KEYSWITCH) Then

                    'get the attribute value
                    strValue = AttributeValues(i)

                    'if the keyswitch is set to RUNP...
                    If (strValue = VAL_KEYSWITCH_RUNP) Then
                        m_chkRUNPbtn.CheckState =
                        System.Windows.Forms.CheckState.Checked
                        m_chkRUNbtn.CheckState =
                        System.Windows.Forms.CheckState.Unchecked
                         m_chkSTOPbtn.CheckState =
        System.Windows.Forms.CheckState.Unchecked
                    'if the keyswitch is set to RUN...
                    ElseIf (strValue = VAL_KEYSWITCH_RUN) Then
                        m_chkRUNPbtn.CheckState =
                        System.Windows.Forms.CheckState.Unchecked
                        m_chkRUNbtn.CheckState =
                        System.Windows.Forms.CheckState.Checked
                         m_chkSTOPbtn.CheckState =
            System.Windows.Forms.CheckState.Unchecked
                    'if the keyswitch is set to STOP...
                    ElseIf (strValue = VAL_KEYSWITCH_STOP) Then
                        m_chkRUNPbtn.CheckState =
                        System.Windows.Forms.CheckState.Unchecked
                m_chkRUNbtn.CheckState =
                System.Windows.Forms.CheckState.Unchecked
                m_chkSTOPbtn.CheckState =
                System.Windows.Forms.CheckState.Checked
            End If
        End If
    Next i

'if it's the led feature...
ElseIf (FeatureName = FEATURE_LED) Then

    'go thru all attributes...
    For i = LBound(AttributeNames) To UBound(AttributeNames)

        'get the attribute name
        strName = AttributeNames(i)

        'if it's the power led attribute...
        If (strName = ATT_LED_POWER) Then

            'get the attribute value
            strValue = AttributeValues(i)

            'if the led is off...
            If (strValue = VAL_LED_OFF) Then
                m_chkON.CheckState =
                System.Windows.Forms.CheckState.Unchecked
            'if it is not off it is on (we don't show blinking LEDs)
            Else
                 m_chkON.CheckState =
    System.Windows.Forms.CheckState.Checked
            End If

        'if it's the batf led attribute...
        ElseIf (strName = ATT_LED_BATF) Then

            'get the attribute value
            strValue = AttributeValues(i)

             'if the led is off...
            If (strValue = VAL_LED_OFF) Then
                m_chkBATF.CheckState =
                System.Windows.Forms.CheckState.Unchecked
            'if it is not off it is on (we don't show blinking LEDs)
            Else
                m_chkBATF.CheckState =
                System.Windows.Forms.CheckState.Checked
             End If

        'if it's the intf led attribute...
        ElseIf (strName = ATT_LED_INTF) Then
            'get the attribute value
            strValue = AttributeValues(i)

            'if the led is off...
            If (strValue = VAL_LED_OFF) Then
                m_chkINTF.CheckState =
                System.Windows.Forms.CheckState.Unchecked
            'if it is not off it is on (we don't show blinking LEDs)
            Else
                m_chkINTF.CheckState =
                System.Windows.Forms.CheckState.Checked
            End If
        'if it's the extf led attribute...
         ElseIf (strName = ATT_LED_EXTF) Then
            'get the attribute value
            strValue = AttributeValues(i)

            'if the led is off...
             If (strValue = VAL_LED_OFF) Then
                m_chkEXTF.CheckState =
                System.Windows.Forms.CheckState.Unchecked
            'if it is not off it is on (we don't show blinking LEDs)
            Else
                m_chkEXTF.CheckState =
                System.Windows.Forms.CheckState.Checked
            End If

        'if it's the busf1 led attribute...
        ElseIf (strName = (ATT_LED_BUSF & "0")) Then
            'get the attribute value
            strValue = AttributeValues(i)

            'if the led is off...
            If (strValue = VAL_LED_OFF) Then
                m_chkBUSF1.CheckState =
                System.Windows.Forms.CheckState.Unchecked
            'if it is not off it is on (we don't show blinking LEDs)
            Else
                m_chkBUSF1.CheckState =
                System.Windows.Forms.CheckState.Checked
            End If
        'if it's the busf2 led attribute...
        ElseIf (strName = (ATT_LED_BUSF & "1")) Then
            'get the attribute value
            strValue = AttributeValues(i)

            'if the led is off...
            If (strValue = VAL_LED_OFF) Then
                m_chkBUSF2.CheckState =
                System.Windows.Forms.CheckState.Unchecked
            'if it is not off it is on (we don't show blinking LEDs)
            Else
                m_chkBUSF2.CheckState =
                System.Windows.Forms.CheckState.Checked
            End If
        'if it's the frce led attribute...
        ElseIf (strName = ATT_LED_FORCE) Then
            'get the attribute value
            strValue = AttributeValues(i)

            'if the led is off...
            If (strValue = VAL_LED_OFF) Then
                m_chkFRCE.CheckState =
                System.Windows.Forms.CheckState.Unchecked
            'if it is not off it is on (we don't show blinking LEDs)
            Else
                m_chkFRCE.CheckState =
                System.Windows.Forms.CheckState.Checked
            End If
        'if it's the run led attribute...
        ElseIf (strName = ATT_LED_RUN) Then
            'get the attribute value
            strValue = AttributeValues(i)

            'if the led is off...
            If (strValue = VAL_LED_OFF) Then
                m_chkRUN.CheckState =
                System.Windows.Forms.CheckState.Unchecked
            'if it is not off it is on (we don't show blinking LEDs)
            Else
                m_chkRUN.CheckState =
                System.Windows.Forms.CheckState.Checked
            End If

        'if it's the stop led attribute...
        ElseIf (strName = ATT_LED_STOP) Then
            'get the attribute value
            strValue = AttributeValues(i)

            'if the led is off...
             If (strValue = VAL_LED_OFF) Then
                m_chkSTOP.CheckState =
                System.Windows.Forms.CheckState.Unchecked
            'if it is not off it is on (we don't show blinking LEDs)
            Else
                m_chkSTOP.CheckState =
                System.Windows.Forms.CheckState.Checked
            End If
         End If
    Next i

 'if it's the plc feature...
 ElseIf (FeatureName = FEATURE_PLC) Then
    'go thru all attributes...
    For i = LBound(AttributeNames) To UBound(AttributeNames)

        'get the attribute name
        strName = AttributeNames(i)

        'if it's the plc attribute...
        If (strName = ATT_PLC) Then

            'get the attribute value
            strValue = AttributeValues(i)

            'if the PLC is created...
            If (strValue = VAL_PLC_CREATED) Then
                Call ActivateControls()
            'if the PLC not running...
            Else
```

```
                    Call DeactivateControls()
                End If
            End If
        Next i

    'if it's the error feature...
    ElseIf (FeatureName = FEATURE_ERROR) Then

        'go thru all attributes...
        For i = LBound(AttributeNames) To UBound(AttributeNames)

            'get the attribute name
            strName = AttributeNames(i)

            'if it's the error id attribute...
            If (strName = ATT_ERROR_ID) Then

                'get the attribute value
                strValue = AttributeValues(i)

                'if it's an error....
                If (strValue <> CStr(Feature.PSERR.PSERR_OKAY)) Then

                    'create message
                    strMessage = "The FeatureProvider returned an error."

                    'show error message
                    MsgBox (strMessage)
               End If
            End If
        Next i

    End If

Error_Renamed:
    End Sub

    Public Sub IFeatureCallback_OnPLCDisconnect(ByVal ErrorID As Integer)
    Implements S7WCUPIntLib.IFeatureCallback.OnPLCDisconnect
        'set the application as disconnected
        Call Me.ConnectionLost()
    End Sub

    Public Sub ClearPlcData()
        'reset
        m_ConnectionNotifyID = -1
        m_KeyswitchNotifyID = -1
        m_LedNotifyID = -1
        m_PlcNotifyID = -1
        m_ErrorNotifyID = -1

        'release the controller management interface
        m_pIfeature = Nothing

        'clear the variants
        m_FeatureNames = Nothing
    End Sub
    Public Function RegisterAllFeatures() As Integer
        On Error GoTo Error_Renamed
        'error
        If m_pIfeature Is Nothing Then
            GoTo Error_Renamed
        End If

        'init
        Dim strFeatureName As String
        Dim i As Integer

      'register to get a notification when the connection to the PLC is lost
        Call m_pIfeature.RegisterForConnectionCheck(Me, m_ConnectionNotifyID,
        ErrorID)

        'error?
        If (ErrorID <> 0) Then
            GoTo Error_Renamed
        End If

        'go thru all features.....
        For i = LBound(m_FeatureNames) To UBound(m_FeatureNames)

            'get the feature name
            strFeatureName = m_FeatureNames(i)

            'if it's one of the features we need...
            If (strFeatureName = FEATURE_KEYSWITCH) Then
                '...register it
                hr = RegisterFeature(strFeatureName, m_KeyswitchNotifyID)
            ElseIf (strFeatureName = FEATURE_LED) Then
                '...register it
                hr = RegisterFeature(strFeatureName, m_LedNotifyID)
            ElseIf (strFeatureName = FEATURE_PLC) Then
                '...register it
                hr = RegisterFeature(strFeatureName, m_PlcNotifyID)
            ElseIf (strFeatureName = FEATURE_ERROR) Then
                '...register it
                hr = RegisterFeature(strFeatureName, m_ErrorNotifyID)
            End If
            'error?
            If (FAILED(hr) = True) Then
                GoTo Error_Renamed
            End If

        Next i

        RegisterAllFeatures = S_OK
        Exit Function
Error_Renamed:
        RegisterAllFeatures = E_FAIL
    End Function

     Public Sub UnregisterAllFeatures()
        On Error GoTo Error_Renamed
        'error
        If m_pIfeature Is Nothing Then
            GoTo Error_Renamed
        End If

        'let's unregister the features => the feature provider stopps sEnding
        notIfications
        If (m_KeyswitchNotifyID <> -1) Then
            Call UnregisterFeature(FEATURE_KEYSWITCH, m_KeyswitchNotifyID)
        End If
        If (m_LedNotifyID <> -1) Then
            Call UnregisterFeature(FEATURE_LED, m_LedNotifyID)
        End If
        If (m_PlcNotifyID <> -1) Then
            Call UnregisterFeature(FEATURE_PLC, m_PlcNotifyID)
        End If
        If (m_ErrorNotifyID <> -1) Then
            Call UnregisterFeature(FEATURE_ERROR, m_ErrorNotifyID)
        End If

        'unregister for connection check
        If (m_ConnectionNotifyID <> -1) Then
            Call m_pIfeature.UnregisterForConnectionCheck(m_ConnectionNotifyID,
            ErrorID)
        End If
Error_Renamed:
    End Sub

    Public Function SetKeyswitch(ByVal Value As String) As Integer
        On Error GoTo Error_Renamed
        'error
        If m_pIfeature Is Nothing Then
            GoTo Error_Renamed
        End If

        'create value array
        Dim AttributeNames(0) As String
        Dim AttributeValues(0) As String
        AttributeNames(0) = ATT_KEYSWITCH
        AttributeValues(0) = Value

      'set the keyswitch
        Call m_pIfeature.SetFeature(FEATURE_KEYSWITCH, AttributeNames,
        AttributeValues, ErrorID)

        'error?
        If (ErrorID <> 0) Then
            GoTo Error_Renamed
        End If

        SetKeyswitch = S_OK
        Exit Function
Error_Renamed:
        SetKeyswitch = E_FAIL
    End Function

    Public Function GetCpuType(ByRef Value As String) As Integer
        On Error GoTo Error_Renamed
        'error
        If m_pIfeature Is Nothing Then
            GoTo Error_Renamed
        End If

        'init
        Value = ""
        Dim AttributeNames As Object
        Dim AttributeValues As Object
        Dim i As Integer

      'get the personality feature attribute values
        Call m_pIfeature.GetFeature(FEATURE_PERSONALITY, AttributeNames,
        AttributeValues, ErrorID)

        'error?
        If (ErrorID <> 0) Then
            GoTo Error_Renamed
        End If
        'loop thru all attributes
        For i = LBound(AttributeNames) To UBound(AttributeNames)

            'if it's the plc type attribute
            If (AttributeNames(i) = ATT_PERSONALITY_TYPE) Then

               'get type
                Value = AttributeValues(i)
                GetCpuType = S_OK
                Exit Function

            End If

        Next i
        GetCpuType = S_False
        Exit Function
Error_Renamed:
        GetCpuType = E_FAIL
    End Function

    Private Function RegisterFeature(ByVal FeatureName As String, ByRef
    NotIficationID As Integer) As Integer
        On Error GoTo Error_Renamed
        'error
        If m_pIfeature Is Nothing Then
            GoTo Error_Renamed
        End If

        'init
        Dim var As Object

        'register the feature to get change notifications
        Call m_pIfeature.RegisterFeatureForChange(Me, FeatureName, var,
        NotIficationID, ErrorID)

        'error?
        If (ErrorID <> 0) Then
            GoTo Error_Renamed
        End If

        RegisterFeature = S_OK
        Exit Function
Error_Renamed:
        RegisterFeature = E_FAIL
    End Function

    Private Function UnregisterFeature(ByVal FeatureName As String, ByVal
    NotIficationID As Integer) As Integer
        On Error GoTo Error_Renamed
        'error
        If m_pIfeature Is Nothing Then
            GoTo Error_Renamed
        End If

        'unregister the feature to cancel notifications
        Call m_pIfeature.UnregisterFeatureForChange(NotIficationID, ErrorID)

        'error?
        If (ErrorID <> 0) Then
            GoTo Error_Renamed
        End If

        UnregisterFeature = S_OK
        Exit Function
Error_Renamed:
        UnregisterFeature = E_FAIL
    End Function

    Public Function ConnectToPlc(ByVal InstanceName As String, ByRef pIfeature
    As S7WCUPIntLib.IFeature, ByRef FeatureNames As Object, ByRef
    AttributeNames As Object) As Integer
        'init
        Dim ConnectStrings As Object
        Dim StartInfos As Object
        Dim strConnectionString As String
        Dim i As Integer

        'get connection strings for all available PLCs
        hr = Browse(ConnectStrings, StartInfos)

        'error?
        If (FAILED(hr) = True) Then
            GoTo Error_Renamed
        End If

        'go thru all returned connection strings and
       'look for the requested instance name
        Dim pos As Integer
        Dim size_Renamed As Integer
        For i = LBound(ConnectStrings) To UBound(ConnectStrings)

            'get connection string
            strConnectionString = ConnectStrings(i)
            'init
            'extract the instance name out of the connection string
            pos = InStr(1, strConnectionString, "\", CompareMethod.Text)
            size_Renamed = InStr(pos + 1, strConnectionString, "\",
            CompareMethod.Text) - pos - 1
            strConnectionString = Mid(strConnectionString, pos + 1,
            size_Renamed)

            'if we have found the correct connection string...
            If (strConnectionString = InstanceName) Then

                'get connection string
                strConnectionString = ConnectStrings(i)

                'init
                pIfeature = Nothing
                FeatureNames = Nothing
                AttributeNames = Nothing
                'connect to the PLC
                Call m_pIPlc.Connect(strConnectionString, pIfeature,
                FeatureNames, AttributeNames, ErrorID)

                'error?
                If (ErrorID <> 0) Then
                    GoTo Error_Renamed
                End If

                ConnectToPlc = S_OK
                Exit Function
            End If
        Next i

Error_Renamed:
        ConnectToPlc = E_FAIL
    End Function

    Private Function Browse(ByRef ConnectStrings As Object, ByRef StartInfos As
    Object) As Integer
        On Error GoTo Error_Renamed

        'clear the arrays
        ConnectStrings = Nothing
        StartInfos = Nothing
        'init
        Dim ErrorID As Integer

        'browse for configured and/or running PLCs
        Call m_pIPlc.Browse(ConnectStrings, StartInfos, ErrorID)

        'error?
        If (ErrorID <> 0) Then
            GoTo Error_Renamed
        End If

        Browse = S_OK
        Exit Function
Error_Renamed:
        Browse = E_FAIL
    End Function
End Class
```

Feature.vb

```
Option Strict Off
Option Explicit On
Module Feature
    '***************************************************************************
    '*   Copyright                                                             *
    '***************************************************************************

    '*--------------------------------------------------------------------------
    '*
    '*   Author      :   Robin Timmermann
    '*   Date        :   03/21/03
    '*
    '*--------------------------------------------------------------------------
    '*
    '*   Description :   feature/attribute/value string defines
    '*
    '*--------------------------------------------------------------------------
    '*
    '*   Modification history:
    '*
    '***************************************************************************


    '---------------------------------------------------------------------------
    '    LED
    '---------------------------------------------------------------------------
    Public Const FEATURE_LED  As String = "LED"

    ' attribute power LED
    Public Const ATT_LED_POWER  As String = "Power"

    ' attribute battery fault LED
    Public Const ATT_LED_BATF As String = "BatteryFault"

    ' attribute internal fault LED
    Public Const ATT_LED_INTF As String = "InternalFault"

    ' attribute external fault LED
    Public Const ATT_LED_EXTF As String = "ExternalFault"

    ' attribute bus fault LED
    Public Const ATT_LED_BUSFAULTCOUNT As String = "BusFaultCount"
    'VAL_LED_BUSFAULTCOUNT                  %d
    Public Const ATT_LED_BUSF As String = "BusFault_"

    ' attribute force LED
    Public Const ATT_LED_FORCE As String = "Force"

    ' attribute run LED
    Public Const ATT_LED_RUN As String = "Run"

    ' attribute stop LED
    Public Const ATT_LED_STOP As String = "Stop"

    ' value LED
```

```
Public Const VAL_LED_ON As String = "ON"
Public Const VAL_LED_OFF As String = "OFF"
Public Const VAL_LED_BLINKING2HZ As String = "Blinking2HZ"
Public Const VAL_LED_BLINKING05HZ As String = "Blinking05HZ"

'---------------------------------------------------------------------------
'    KeySwitch
'---------------------------------------------------------------------------
Public Const FEATURE_KEYSWITCH As String = "KeySwitch"

' attribute KeySwitch
Public Const ATT_KEYSWITCH As String = "Value"
Public Const VAL_KEYSWITCH_MRES As String = "MRES"
Public Const VAL_KEYSWITCH_STOP As String = "STOP"
Public Const VAL_KEYSWITCH_RUN As String = "RUN"
Public Const VAL_KEYSWITCH_RUNP As String = "RUNP"

' attribute force coldstart
Public Const ATT_KEYSWITCH_FORCECOLDSTART As String = "ForceColdstart"
'Public Const VAL_ON = "On"
Public Const VAL_OFF As String = "Off"

'---------------------------------------------------------------------------
'    PLCInstance
'---------------------------------------------------------------------------
Public Const FEATURE_PLCINSTANCE As String = "PLCInstance"

' attribute PLCINSTANCE
Public Const ATT_PLCINSTANCE As String = "Value"
Public Const VAL_PLCINSTANCE_CREATE As String = "Create"
Public Const VAL_PLCINSTANCE_SHUTDOWN As String = "Shutdown"

'---------------------------------------------------------------------------
'    PLCPower
'---------------------------------------------------------------------------
Public Const FEATURE_PLCPOWER As String = "PLCPower"

' attribute PLCPower state
Public Const ATT_PLCPOWER As String = "Value"
Public Const VAL_ON As String = "On"
'Public Const VAL_OFF = "Off"

'---------------------------------------------------------------------------
'    PLC
'---------------------------------------------------------------------------
Public Const FEATURE_PLC As String = "PLC"

' attribute PLC
Public Const ATT_PLC As String = "Value"
Public Const VAL_PLC_CREATED As String = "Created"
Public Const VAL_PLC_SHUTDOWNED As String = "Shutdowned"
Public Const VAL_PLC_NOTAVAILABLE As String = "NotAvailable"

'---------------------------------------------------------------------------
'    StartAtBoot
'---------------------------------------------------------------------------
Public Const FEATURE_STARTATBOOT As String = "StartAtBoot"
```

```
' attribute Start at boot
Public Const ATT_STARTATBOOT As String = "Value"
'Public Const VAL_ON = "On"
'Public Const VAL_OFF = "Off"

'---------------------------------------------------------------------------
'    AutoStart
'---------------------------------------------------------------------------
Public Const FEATURE_AUTOSTART As String = "AutoStart"

' attribute Autostart
Public Const ATT_AUTOSTART As String = "Value"
'Public Const VAL_ON = "On"
'Public Const VAL_OFF = "Off"

'---------------------------------------------------------------------------
'    AutoLoad
'---------------------------------------------------------------------------
Public Const FEATURE_AUTOLOAD As String = "AutoLoad"

' attribute Autoload enabled
Public Const ATT_AUTOLOAD As String = "Value"
'Public Const VAL_ON = "On"
'Public Const VAL_OFF = "Off"

' attribute Keyswitch state after Autoload
Public Const ATT_AUTOLOAD_KEYSWITCH As String = "KeySwitch"
Public Const VAL_AUTOLOAD_KS_STOP As String = "STOP"
Public Const VAL_AUTOLOAD_KS_RUN As String = "RUN"
Public Const VAL_AUTOLOAD_KS_RUNP As String = "RUNP"

' attribute Buffer for Autoload
Public Const ATT_AUTOLOAD_BUFFER As String = "Buffer"
'Public Const VAL_AUTOLOAD_BUFFER             %s

' attribute Buffersize for Autoload
Public Const ATT_AUTOLOAD_BUFFERSIZE As String = "BufferSize"
'Public Const VAL_AUTOLOAD_BUFFERSIZE         %d

' attribute target filename for Autoload
Public Const ATT_AUTOLOAD_TARGETFILE As String = "TargetFile"
'Public Const VAL_AUTOLOAD_TARGETFILE         %s

'---------------------------------------------------------------------------
'    Security
'---------------------------------------------------------------------------
Public Const FEATURE_SECURITY As String = "Security"

' attribute actual Password
Public Const ATT_SECURITY_ACTPASSWORD As String = "Password"
'Public Const VAL_SECURITY_ACTPASSWORD        %s scrammbled

' attribute new Password
Public Const ATT_SECURITY_NEWPASSWORD As String = "NewPassword"
'Public Const VAL_SECURITY_NEWPASSWORD        %s scrammbled
```

```
' attribute Password check
Public Const ATT_SECURITY_PASSWORDCHECK As String = "Check"
Public Const VAL_SECURITY_PASSWORDCHECK_PASS As String = "Passed"
Public Const VAL_SECURITY_PASSWORDCHECK_FAILED As String = "Failed"

' attribute security level
Public Const ATT_SECURITY_LEVEL As String = "Level"
Public Const VAL_SECURITY_LEVELPASSWORD As String = "Password"
Public Const VAL_SECURITY_LEVELPASSWORDDISABLED As String =
"PasswordDisabled"
Public Const VAL_SECURITY_LEVELCONFIRMATION As String = "Confirmation"
Public Const VAL_SECURITY_LEVELNONE As String = "None"

' attribute Password Prompt Interval hours
Public Const ATT_SECURITY_INTERVALHOURS As String = "hInterval"
'Public Const VAL_SECURITY_INTERVALHOURS      %d

' attribute Password Prompt Interval minutes
Public Const ATT_SECURITY_INTERVALMINUTES As String = "mInterval"
'Public Const VAL_SECURITY_INTERVALMINUTES    %d

'---------------------------------------------------------------------------
'    FMR
'---------------------------------------------------------------------------
Public Const FEATURE_FMR As String = "FMR"

'---------------------------------------------------------------------------
'    MemoryCard file
'---------------------------------------------------------------------------
Public Const FEATURE_MCF As String = "MemoryCardFile"

' attribute MemoryCard file Buffer
Public Const ATT_MCF_BUFFER As String = "Buffer"
'VAL_MCF_BUFFER                          %s

' attribute MemoryCard file Size
Public Const ATT_MCF_SIZE As String = "Size"
'VAL_MCF_SIZE                            %d

'---------------------------------------------------------------------------
'    Priority
'---------------------------------------------------------------------------
Public Const FEATURE_PRIORITY As String = "Priority"

' attribute Priority value
Public Const ATT_PRIORITY As String = "Value"
'Public Const VAL_PRIORITY                    %d

Public Const ATT_PRIORITY_LOWERLIMIT As String = "LowerLimit"
'Public Const VAL_PRIORITY_LOWERLIMIT         %d

Public Const ATT_PRIORITY_UPPERLIMIT As String = "UpperLimit"
'Public Const VAL_PRIORITY_UPPERLIMIT         %d

Public Const ATT_NORMAL_PRIORITY As String = "Normal"
'Public Const VAL_NORMAL_PRIORITY             %d
```

```
Public Const ATT_CRITICAL_PRIORITY As String = "Critical"
'Public Const VAL_CRITICAL_PRIORITY                %d

'---------------------------------------------------------------------------
'    Minimum Sleep Time
'---------------------------------------------------------------------------
Public Const FEATURE_MINSLEEPTIME As String = "MinSleepTime"

' attribute Minimum SleepTime value
Public Const ATT_MINSLEEPTIME As String = "Value"
'Public Const VAL_MINSLEEPTIME                %d

Public Const ATT_MINSLEEPTIME_LOWERLIMIT As String = "LowerLimit"
'Public Const VAL_MINSLEEPTIME_LOWERLIMIT     %d

Public Const ATT_MINSLEEPTIME_UPPERLIMIT As String = "UpperLimit"
'Public Const VAL_MINSLEEPTIME_UPPERLIMIT     %d

'---------------------------------------------------------------------------
'    Minimum Cycle Time
'---------------------------------------------------------------------------
Public Const FEATURE_MINCYCLETIME As String = "MinCycleTime"

' attribute Minimum CycleTime value
Public Const ATT_MINCYCLETIME As String = "Value"
'Public Const VAL_MINCYCLETIME                %d

Public Const ATT_MINCYCLETIME_LOWERLIMIT As String = "LowerLimit"
'Public Const VAL_MINSLEEPTIME_LOWERLIMIT     %d

Public Const ATT_MINCYCLETIME_UPPERLIMIT As String = "UpperLimit"
'Public Const VAL_MINCYCLETIME_UPPERLIMIT     %d

'---------------------------------------------------------------------------
'    CPU Usage
'---------------------------------------------------------------------------
Public Const FEATURE_CPUUSAGE As String = "Usage"

' attribute Usage count
Public Const ATT_PCUSAGE As String = "PC"
'VAL_PCUSAGE                             %d [0..100%]

' attribute PLC Usage
Public Const ATT_PLCUSAGE As String = "PLC"
'Public Const VAL_PLCUSAGE                    %d [0..100%]

' attribute Usage count
Public Const ATT_CPUUSAGE_COUNT As String = "CPUCount"
'VAL_CPUUSAGE_COUNT                      %d

' attribute CPU Usage values
Public Const ATT_CPUUSAGE As String = "CPU_"
'Public Const VAL_CPUUSAGE                    %d [0..100%]

'---------------------------------------------------------------------------
'    Timing
'---------------------------------------------------------------------------
```

```
Public Const FEATURE_TIMING As String = "Timing"

' attribute Timing Upper Limit
Public Const ATT_TIMING_UPPERLIMIT As String = "UpperLimit"
'VAL_TIMING_UPPERLIMIT                       %d

' attribute CycleTime count
Public Const ATT_TIMING_CYCLETIMECOUNT As String = "CycleTimeCount"
'Public Const VAL_TIMING_CYCLETIMECOUNT       %d

' attribute CycleTime buffer
Public Const ATT_TIMING_CYCLETIMEBUFFER As String = "CycleTimeBuffer"
'Public Const VAL_TIMING_CYCLETIMEBUFFER      %d %d...

' attribute CycleTime minimum
Public Const ATT_TIMING_CYCLETIMEMIN As String = "CycleTimeMin"
'Public Const VAL_TIMING_CYCLETIMEMIN         %d

' attribute CycleTime maximum
Public Const ATT_TIMING_CYCLETIMEMAX As String = "CycleTimeMax"
'Public Const VAL_TIMING_CYCLETIMEMAX         %d

' attribute CycleTime average
Public Const ATT_TIMING_CYCLETIMEAVE As String = "CycleTimeAverage"
'Public Const VAL_TIMING_CYCLETIMEAVE         %d

' attribute CycleTime last
Public Const ATT_TIMING_CYCLETIMELAST As String = "CycleTimeLast"
'Public Const VAL_TIMING_CYCLETIMELAST        %d

' attribute ExecTime minimum
Public Const ATT_TIMING_EXECTIMEMIN As String = "ExecTimeMin"
'Public Const VAL_TIMING_EXECTIMEMIN          %d

' attribute ExecTime maximum
Public Const ATT_TIMING_EXECTIMEMAX As String = "ExecTimeMax"
'Public Const VAL_TIMING_EXECTIMEMAX          %d

' attribute ExecTime average
Public Const ATT_TIMING_EXECTIMEAVE As String = "ExecTimeAverage"
'Public Const VAL_TIMING_EXECTIMEAVE          %d

' attribute ExecTime last
Public Const ATT_TIMING_EXECTIMELAST As String = "ExecTimeLast"
'Public Const VAL_TIMING_EXECTIMELAST         %d

' attribute SleepIntervalCounter
Public Const ATT_TIMING_SLEEPINTERVALCOUNTER As String =
"SleepIntervalCounter"
'VAL_TIMING_SLEEPINTERVALCOUNTER         %d

' attribute Clear
Public Const ATT_TIMING_CLEAR As String = "Clear"
'Public Const VAL_TIMING_CLEAR                empty

'---------------------------------------------------------------------------
'    OBExecution
```

```
'--------------------------------------------------------------------------
Public Const FEATURE_OBEXEC As String = "OBExecution"

' attribute WakeInterval
Public Const ATT_OBEXEC_WAKEINTERVAL As String = "WakeInterval"
'VAL_OBEXEC_WAKEINTERVAL                  %d

' attribute SleepInterval
Public Const ATT_OBEXEC_SLEEPINTERVAL As String = "SleepInterval"
'VAL_OBEXEC_SLEEPINTERVAL                 %d

' attribute default WakeInterval
Public Const ATT_OBEXEC_DEFAULTWAKEINTERVAL As String =
"DefaultWakeInterval"
'VAL_OBEXEC_DEFAULTWAKEINTERVAL           %d

' attribute default SleepInterval
Public Const ATT_OBEXEC_DEFAULTSLEEPINTERVAL As String =
"DefaultSleepInterval"
'VAL_OBEXEC_DEFAULTSLEEPINTERVAL          %d

' attribute UpperLimit maximum execution load
Public Const ATT_OBEXEC_UPPERLIMIT As String = "UpperLimit"
'VAL_OBEXEC_UPPERLIMIT                    %d [0..100%]

' attribute Lowerimit maximum execution load
Public Const ATT_OBEXEC_LOWERLIMIT As String = "LowerLimit"
'VAL_OBEXEC_LOWERLIMIT                    %d [0..100%]

'--------------------------------------------------------------------------
'    Diagnostic Language
'--------------------------------------------------------------------------
Public Const FEATURE_DIAGLANGUAGE As String = "DiagnosticLanguage"

' attribute Language count
Public Const ATT_DIAGLANGUAGE_COUNT As String = "Count"
'VAL_DIAGLANGUAGE_COUNT                   %d

' attribute Language
Public Const ATT_DIAGLANGUAGE As String = "Language_"
Public Const VAL_LANGUAGE_GERMAN As String = "GERMAN"
Public Const VAL_LANGUAGE_ENGLISH As String = "ENGLISH"
Public Const VAL_LANGUAGE_FRENCH As String = "FRENCH"
Public Const VAL_LANGUAGE_ITALIAN As String = "ITALIAN"
Public Const VAL_LANGUAGE_SPANISH As String = "SPANISH"
Public Const VAL_LANGUAGE_JAPANESE As String = "JAPANESE"
Public Const VAL_LANGUAGE_CHINESE As String = "CHINESE"

'--------------------------------------------------------------------------
'    Diagnostic Information
'--------------------------------------------------------------------------
Public Const FEATURE_DIAGNOSTIC As String = "Diagnostic"

' attribute Language
Public Const ATT_DIAG_LANGUAGE As String = "Language"
'VAL_DIAG_LANGUAGE                        %s
```

```
' attribute Diagnostic entry count
Public Const ATT_DIAG_COUNT As String = "Count"
'VAL_DIAG_COUNT                          %d

' attribute Time
Public Const ATT_DIAG_TIME As String = "Time_"
'VAL_DIAG_TIME                           %s

' attribute Date
Public Const ATT_DIAG_DATE As String = "Date_"
'VAL_DIAG_DATE                           %s

' attribute Event short text
Public Const ATT_DIAG_EVENTSHORT As String = "EventShort_"
'VAL_DIAG_EVENTSHORT                     %s

' attribute Event long text
Public Const ATT_DIAG_EVENTLONG As String = "EventLong_"
'VAL_DIAG_EVENTLONG                      %s

' attribute Event ID
Public Const ATT_DIAG_EVENTID As String = "EventID_"
'VAL_DIAG_EVENTID                        %s

' attribute Event hex text
Public Const ATT_DIAG_EVENTHEX As String = "EventHex_"
'VAL_DIAG_EVENTHEX                       %s

'--------------------------------------------------------------------------
'    Personality
'--------------------------------------------------------------------------
Public Const FEATURE_PERSONALITY As String = "Personality"

' attribute instance name
Public Const ATT_PERSONALITY_NAME As String = "Name"
'VAL_PERSONALITY_NAME                    %s

' attribute CPU type
Public Const ATT_PERSONALITY_TYPE As String = "Type"
'ATT_PERSONALITY_TYPE                    %s

' attribute product code type
Public Const ATT_PERSONALITY_PRODUCTCODE As String = "ProductCode"
'ATT_PERSONALITY_PRODUCTCODE             %s

' attribute SW release
Public Const ATT_PERSONALITY_SW_RELEASE As String = "SWRelease"
'ATT_PERSONALITY_SW_RELEASE              %s

' attribute FW release
Public Const ATT_PERSONALITY_FW_RELEASE As String = "FWRelease"
'ATT_PERSONALITY_FW_RELEASE              %s

' attribute HW release
Public Const ATT_PERSONALITY_HW_RELEASE As String = "HWRelease"
'ATT_PERSONALITY_HW_RELEASE              %s
```

```
' attribute SLOT Number
Public Const ATT_PERSONALITY_SLOT_NUMBER As String = "Slot"
'ATT_PERSONALITY_SLOT_NUMBER             %s

' attribute RACK Number
Public Const ATT_PERSONALITY_RACK_NUMBER As String = "Rack"
'ATT_PERSONALITY_RACK_NUMBER             %s

' attribute OWNER Info
Public Const ATT_PERSONALITY_OWNER_INFO As String = "Owner"
'ATT_PERSONALITY_OWNER_INFO              %s

' attribute RACK Number
Public Const ATT_PERSONALITY_COMPANY_INFO As String = "Company"
'ATT_PERSONALITY_COMPANY_INFO            %s

'--------------------------------------------------------------------------
'    Error
'--------------------------------------------------------------------------
Public Const FEATURE_ERROR As String = "Error"

' attribute error identification
Public Const ATT_ERROR_ID As String = "ID"
'VAL_ERROR_ID                            %d

' valid error identifications
Public Enum PSERR
    PSERR_OKAY = 0
    PSERR_NO_MEMORY
    PSERR_ARCHIVE_NOT_VALID_IN_RUN
    PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU
    PSERR_RESTORE_CANNOT_LINKIN_BLOCK
    PSERR_RESTORE_NOT_VALID_IN_RUN
    PSERR_RESTORE_FILE_INVALID
    PSERR_INIT_EDBSERVER
    PSERR_INIT_PDH
    PSERR_KEYSWITCH_NOT_ALLOWED_IN_MCF_OP
    PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU_PASSWORD_PROTECTED
End Enum

'--------------------------------------------------------------------------
'    HelpInfo
'--------------------------------------------------------------------------
Public Const FEATURE_CONTROLLER_HELP As String = "ControllerHelp"

' attribute host
Public Const ATT_HELP_HOST As String = "Host"
'VAL_HELP_HOST                           %s

' attribute helpsystem
Public Const ATT_HELP_SYSTEM As String = "HelpSystem"
Public Const VAL_HELP_SYSTEM_WIN As String = "WinHelp"
Public Const VAL_HELP_SYSTEM_WEB As String = "WebHelp"
Public Const VAL_HELP_SYSTEM_HTML As String = "HTMLHelp"

' attribute document help directory; includes drive letter
Public Const ATT_HELP_DIR As String = "HelpDir"
```

```
    'VAL_HELP_DIR                            %s

    ' attribute Language count
    Public Const ATT_HELPLANGUAGE_COUNT As String = "Count"
    'VAL_HELPLANGUAGE_COUNT                  %d

    ' attribute Language
    Public Const ATT_HELPLANGUAGE As String = "Language_"
    '
    'Public Const VAL_LANGUAGE_GERMAN = "GERMAN"
    'Public Const VAL_LANGUAGE_ENGLISH = "ENGLISH"
    'Public Const VAL_LANGUAGE_FRENCH = "FRENCH"
    'Public Const VAL_LANGUAGE_ITALIAN = "ITALIAN"
    'Public Const VAL_LANGUAGE_SPANISH = "SPANISH"
    'Public Const VAL_LANGUAGE_JAPANESE = "JAPANESE"
    'Public Const VAL_LANGUAGE_CHINESE = "CHINESE"


    '--------------------------------------------------------------------------
    '    CPU Language
    '--------------------------------------------------------------------------
    Public Const FEATURE_CPULANGUAGE As String = "CPULanguage"

    ' attribute Language
    Public Const ATT_CPULANGUAGE_CURRENT As String = "CurrentLanguage"
    '
    'Public Const VAL_LANGUAGE_GERMAN = "GERMAN"
    'Public Const VAL_LANGUAGE_ENGLISH = "ENGLISH"
    'Public Const VAL_LANGUAGE_FRENCH = "FRENCH"
    'Public Const VAL_LANGUAGE_ITALIAN = "ITALIAN"
    'Public Const VAL_LANGUAGE_SPANISH = "SPANISH"
    'Public Const VAL_LANGUAGE_JAPANESE = "JAPANESE"
    'Public Const VAL_LANGUAGE_CHINESE = "CHINESE"


    ' attribute Language count
    Public Const ATT_CPULANGUAGE_COUNT As String = "Count"
    'VAL_CPULANGUAGE_COUNT                   %d

    ' attribute Language
    Public Const ATT_CPULANGUAGE As String = "Language_"
    '
    'Public Const VAL_LANGUAGE_GERMAN = "GERMAN"
    'Public Const VAL_LANGUAGE_ENGLISH = "ENGLISH"
    'Public Const VAL_LANGUAGE_FRENCH = "FRENCH"
    'Public Const VAL_LANGUAGE_ITALIAN = "ITALIAN"
    'Public Const VAL_LANGUAGE_SPANISH = "SPANISH"
    'Public Const VAL_LANGUAGE_JAPANESE = "JAPANESE"
    'Public Const VAL_LANGUAGE_CHINESE = "CHINESE"


    '---------------------------------------------------------------------------
    '    PC/PG Interface
    '---------------------------------------------------------------------------
    Public Const FEATURE_PCPGINTERFACE  As String = "PC_PG_Interface"
End Module
```

StartModule.vb

```
Option Strict Off
Option Explicit On
Module StartModule
    Public frmDemoDlg As DemoDlg

    Public Sub Main()
        'at this point we create and display the
        'panel dialog
        Set frmDemoDlg = New DemoDlg
        frmDemoDlg.Show
    End Sub
End Module
```

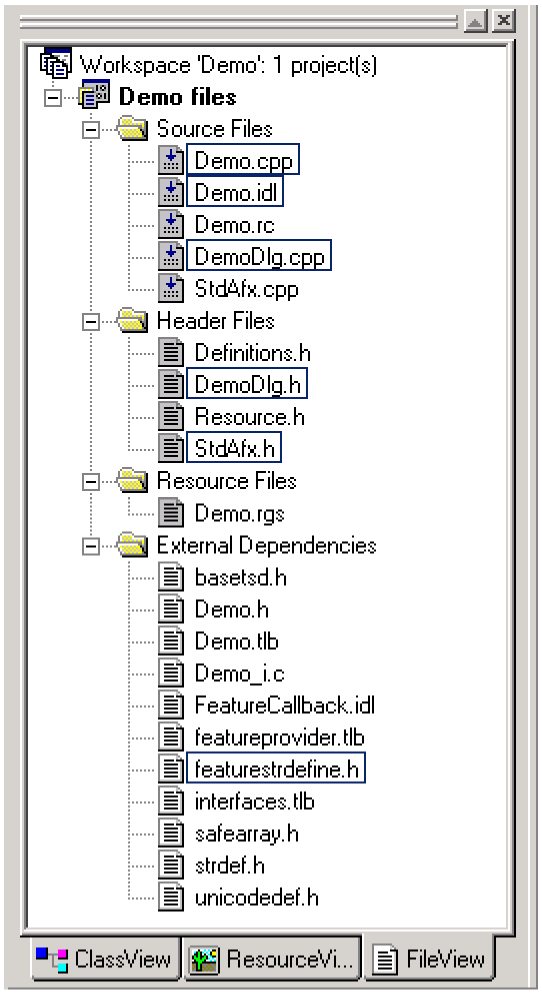
Visual C++ 6.0 and .NET Sample Programs

Introduction to the Visual C++ 6.0 Demo Panel Program

As described in Introducing the Sample Programs, WinAC ODK installs a Visual C++ 6.0 implementation of a control panel that can interact with a WinAC controller. The Visual 6.0 C++ implementation is an ATL COM AppWizard project. This project is located in the ...\Program Files\Siemens\WinAC\ODK\Examples\CMI\Demo_Panel_Cpp directory.

> **Note**
> The Visual C++ 6.0 and Visual C++ .NET source files are the same. Only the project workspaces differ.

You can click one of the indicated files in the C++ File View picture to access its code listing (online help only):



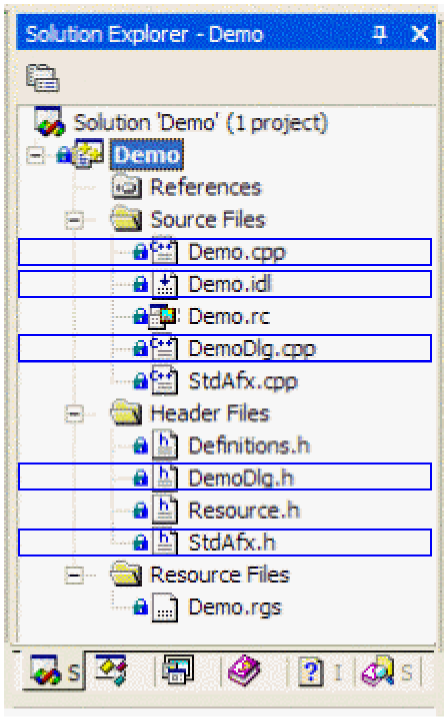
```
Workspace 'Demo': 1 project(s)
  Demo files
    Source Files
      Demo.cpp
      Demo.idl
      Demo.rc
      DemoDlg.cpp
      StdAfx.cpp
    Header Files
      Definitions.h
      DemoDlg.h
      Resource.h
      StdAfx.h
    Resource Files
      Demo.rgs
    External Dependencies
      basetsd.h
      Demo.h
      Demo.tlb
      Demo_i.c
      FeatureCallback.idl
      featureprovider.tlb
      featurestrdefine.h
      interfaces.tlb
      safearray.h
      strdef.h
      unicodedef.h
```

Introduction to the Visual C++ .NET Demo Panel Program

As described in Introducing the Sample Programs, WinAC ODK installs a Visual C++ .NET implementation of a control panel that can interact with a WinAC controller. The Visual C++ .NET implementation is an ATL project. This project is located in the ...\Program Files\Siemens\WinAC\ODK\Examples\CMI\Demo_Panel_Cpp_NET directory.

> **Note**
> The Visual C++ 6.0 and Visual C++ .NET source files are the same. Only the project workspaces differ.

You can click one of the indicated files in the C++ File View picture to access its code listing (online help only):



Note
StdAfx.h includes featureStrDefine.h.

Source Files

Demo.cpp

```
// Demo.cpp : Implementation of WinMain

// Note: Proxy/Stub Information
//      To build a separate proxy/stub DLL,
//      run nmake -f Demops.mk in the project directory.

#include "stdafx.h"
#include "resource.h"
#include <initguid.h>
#include "Demo.h"
#include "Demo_i.c"
#include "DemoDlg.h"


const DWORD dwTimeOut = 5000; // time for EXE to be idle before shutting down
const DWORD dwPause = 1000; // time to wait for threads to finish up

// Passed to CreateThread to monitor the shutdown event
static DWORD WINAPI MonitorProc(void* pv)
{
    CExeModule* p = (CExeModule*)pv;
    p->MonitorShutdown();
    return 0;
}

LONG CExeModule::Unlock()
{
    LONG l = CComModule::Unlock();
    if (l == 0)
    {
        bActivity = true;
        SetEvent(hEventShutdown); // tell monitor that we transitioned to zero
    }
    return l;
}

//Monitors the shutdown event
void CExeModule::MonitorShutdown()
{
    while (1)
    {
        WaitForSingleObject(hEventShutdown, INFINITE);
        DWORD dwWait=0;
        do
        {
            bActivity = false;
            dwWait = WaitForSingleObject(hEventShutdown, dwTimeOut);
        } while (dwWait == WAIT_OBJECT_0);
        // timed out
        if (!bActivity && m_nLockCnt == 0) // if no activity let's really bail
        {
#if _WIN32_WINNT >= 0x0400 & defined(_ATL_FREE_THREADED)
            CoSuspendClassObjects();
            if (!bActivity && m_nLockCnt == 0)
#endif
                break;
        }
    }
    CloseHandle(hEventShutdown);
    PostThreadMessage(dwThreadID, WM_QUIT, 0, 0);
}

bool CExeModule::StartMonitor()
{
    hEventShutdown = CreateEvent(NULL, false, false, NULL);
    if (hEventShutdown == NULL)
        return false;
    DWORD dwThreadID;
    HANDLE h = CreateThread(NULL, 0, MonitorProc, this, 0, &dwThreadID);
    return (h != NULL);
}

CExeModule _Module;

BEGIN_OBJECT_MAP(ObjectMap)
      OBJECT_ENTRY(CLSID_DemoDlg, CDemoDlg)
END_OBJECT_MAP()

LPCTSTR FindOneOf(LPCTSTR p1, LPCTSTR p2)
{
    while (p1 != NULL && *p1 != NULL)
    {
        LPCTSTR p = p2;
        while (p != NULL && *p != NULL)
        {
            if (*p1 == *p)
                return CharNext(p1);
            p = CharNext(p);
        }
        p1 = CharNext(p1);
    }
    return NULL;
}

/////////////////////////////////////////////////////////////////////////////

extern "C" int WINAPI _tWinMain(HINSTANCE hInstance,
    HINSTANCE /*hPrevInstance*/, LPTSTR lpCmdLine, int /*nShowCmd*/)
{
    lpCmdLine = GetCommandLine(); //this line necessary for _ATL_MIN_CRT

#if _WIN32_WINNT >= 0x0400 & defined(_ATL_FREE_THREADED)
    HRESULT hRes = CoInitializeEx(NULL, COINIT_MULTITHREADED);
#else
    HRESULT hRes = CoInitialize(NULL);
#endif
    _ASSERTE(SUCCEEDED(hRes));
    _Module.Init(ObjectMap, hInstance, &LIBID_DEMOLib);
    _Module.dwThreadID = GetCurrentThreadId();
    TCHAR szTokens[] = _T("-/");
    int nRet = 0;
    BOOL bRun = TRUE;
    LPCTSTR lpszToken = FindOneOf(lpCmdLine, szTokens);
    while (lpszToken != NULL)
    {
        if (lstrcmpi(lpszToken, _T("UnregServer"))==0)
        {
            _Module.UpdateRegistryFromResource(IDR_DEMO, FALSE);
            nRet = _Module.UnregisterServer(TRUE);
            bRun = FALSE;
            break;
        }
        if (lstrcmpi(lpszToken, _T("RegServer"))==0)
        {
            _Module.UpdateRegistryFromResource(IDR_DEMO, TRUE);
            nRet = _Module.RegisterServer(TRUE);
            bRun = FALSE;
            break;
        }
        lpszToken = FindOneOf(lpszToken, szTokens);
    }

    if (bRun)
    {
        _Module.StartMonitor();
#if _WIN32_WINNT >= 0x0400 & defined(_ATL_FREE_THREADED)
        hRes = _Module.RegisterClassObjects(CLSCTX_LOCAL_SERVER,
            REGCLS_MULTIPLEUSE | REGCLS_SUSPENDED);
        _ASSERTE(SUCCEEDED(hRes));
        hRes = CoResumeClassObjects();
#else
        hRes = _Module.RegisterClassObjects(CLSCTX_LOCAL_SERVER,
            REGCLS_MULTIPLEUSE);
#endif
        _ASSERTE(SUCCEEDED(hRes));

            // init
            MSG message;
            CDemoDlg* pPanelDlg = NULL;
            IFeatureCallbackPtr pICallback;

            // create our COM panel dialog
            pICallback.CreateInstance(__uuidof(DemoDlg));
            if(pICallback.GetInterfacePtr())
            {
                  // display the panel dialog
                  pPanelDlg = (CDemoDlg*)pICallback.GetInterfacePtr();
                  pPanelDlg->Create(NULL);
                  pPanelDlg->ShowWindow(SW_SHOWNORMAL);

                  // start the message pump
                  while(GetMessage(&message, 0, 0, 0))
                  {
                        // the dialog is closed, so stop the message pump
                        if(message.message == WM_QUIT || message.message ==
WM_NULL)
```

```
                              break;

                        TranslateMessage(&message);
                        DispatchMessage(&message);
                  }

                  // let's destroy the panel dialog
            #ifdef _DEBUG
                  pPanelDlg->m_bModal = FALSE;
            #endif
                  pPanelDlg->DestroyWindow();

                  // release the interface pointer to the callback interface
                  pICallback.Release();
            }

        _Module.RevokeClassObjects();
        Sleep(dwPause); //wait for any threads to finish
    }

    _Module.Term();
    CoUninitialize();
    return nRet;
}
```

Demo.idl

```
// Demo.idl : IDL source for Demo.dll
//

// This file will be processed by the MIDL tool to
// produce the type library (Demo.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";

[
      uuid(75D94BF9-B83B-4101-A255-1C3206A1394B),
      version(1.0),
      helpstring("Demo 1.0 Type Library")
]
library DEMOLib
{
      importlib("stdole32.tlb");
      importlib("stdole2.tlb");

      #include "FeatureCallback.idl"

      [
            uuid(8FDA52F5-A330-409D-9040-B9F95DA3B4A1),
            helpstring("DemoDlg Class")
      ]
      coclass DemoDlg
      {
            [default] interface IFeatureCallback;
      };
};
```

DemoDlg.cpp

```
// DemoDlg.cpp : Implementation of CDemoDlg
#include "stdafx.h"
#include "DemoDlg.h"

/////////////////////////////////////////////////////////////////////////////
// CDemoDlg

LRESULT CDemoDlg::OnInitDialog(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL&
bHandled)
{
      // init and/or attach our controls
      m_chkON.Init(this->m_hWnd, IDC_ON);
      m_chkBATF.Init(this->m_hWnd, IDC_BATF);
      m_chkINTF.Init(this->m_hWnd, IDC_INTF);
      m_chkEXTF.Init(this->m_hWnd, IDC_EXTF);
      m_chkBUSF1.Init(this->m_hWnd, IDC_BUSF1);
      m_chkBUSF2.Init(this->m_hWnd, IDC_BUSF2);
      m_chkFRCE.Init(this->m_hWnd, IDC_FRCE);
      m_chkRUN.Init(this->m_hWnd, IDC_RUN);
      m_chkSTOP.Init(this->m_hWnd, IDC_STOP);
      m_chkRUNPbtn.Init(this->m_hWnd, IDC_RUNP_CHK);
      m_chkRUNbtn.Init(this->m_hWnd, IDC_RUN_CHK);
      m_chkSTOPbtn.Init(this->m_hWnd, IDC_STOP_CHK);
      m_btnConnect.Init(this->m_hWnd, IDC_CONNECT_BTN);
      m_btnType.Init(this->m_hWnd, IDC_GET_TYPE);
      m_btnRUNP.Init(this->m_hWnd, IDC_RUNP_BTN);
      m_btnRUN.Init(this->m_hWnd, IDC_RUN_BTN);
      m_btnSTOP.Init(this->m_hWnd, IDC_STOP_BTN);
      m_btnMRES.Init(this->m_hWnd, IDC_MRES_BTN);
      m_edtInstanceName.Init(this->m_hWnd, IDC_INSTANCENAME);
      m_edtType.Init(this->m_hWnd, IDC_TYPE);

      // set default instance name
      m_edtInstanceName.SetWindowText(_T("WinLC"));

      // clear plc type info
      m_edtType.SetWindowText(_T(""));

      // set the initial connect button text
      m_btnConnect.SetWindowText(BTN_TXT_CONNECT);

      // disable all panel controls
      this->DeactivateControls();

      return 1;  // Let the system set the focus
}

LRESULT CDemoDlg::OnCancel(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled)
{
      // disconnect
      this->Disconnect();

#ifdef _DEBUG
      m_bModal = TRUE;
```

```
#endif

      EndDialog(wID);
      return 0;
}

STDMETHODIMP CDemoDlg::OnFeatureChanged(BSTR FeatureName, VARIANT Context, long
NotificationID, VARIANT AttributeNames, VARIANT AttributeValues)
{
      // init
      wstring strName;
      wstring strValue;
      CSafeStringArray saNames;
      CSafeStringArray saValues;
      wstring strFeatureName;

      // get feature name
      strFeatureName = FeatureName;

      // get the arrays
      saNames.Attach(AttributeNames);
      saValues.Attach(AttributeValues);

      // if it's the keyswitch feature...
      if(!strFeatureName.compare(wstring(FEATURE_KEYSWITCH)))
      {
            // go thru all attributes...
            for(UINT i=0; i<saNames.GetSize(); i++)
            {
                  // get the attribute name
                  strName = saNames(i);

                  // if it's the keyswitch attribute...
                  if(!strName.compare(wstring(ATT_KEYSWITCH)))
                  {
                        // get the attribute value
                        strValue = saValues(i);

                        // if the keyswitch is set to RUNP...
                        if(!strValue.compare(wstring(VAL_KEYSWITCH_RUNP)))
                        {
                              m_chkRUNPbtn.SetCheck(1);
                              m_chkRUNbtn.SetCheck(0);
                              m_chkSTOPbtn.SetCheck(0);
                       }
                        // if the keyswitch is set to RUN...
                        else if(!strValue.compare(wstring(VAL_KEYSWITCH_RUN)))
                        {
                              m_chkRUNPbtn.SetCheck(0);
                              m_chkRUNbtn.SetCheck(1);
                              m_chkSTOPbtn.SetCheck(0);
                       }
                        // if the keyswitch is set to STOP...
                        else if(!strValue.compare(wstring(VAL_KEYSWITCH_STOP)))
                        {
                              m_chkRUNPbtn.SetCheck(0);
                              m_chkRUNbtn.SetCheck(0);
                              m_chkSTOPbtn.SetCheck(1);
                        }
                  }
            }
      }
      // if it's the led feature...
      else if(!strFeatureName.compare(wstring(FEATURE_LED)))
      {
            // go thru all attributes...
            for(UINT i=0; i<saNames.GetSize(); i++)
            {
                  // get the attribute name
                  strName = saNames(i);

                  // if it's the power led attribute...
                  if(!strName.compare(wstring(ATT_LED_POWER)))
                  {
                        // get the attribute value
                        strValue = saValues(i);

                        // if the led is off...
                        if(!strValue.compare(wstring(VAL_LED_OFF)))
                              m_chkON.SetCheck(0);
                        // if it is not off it is on (we don't show blinking
LEDs)
                        else
                              m_chkON.SetCheck(1);
                  }
                  // if it's the batf led attribute...
                  else if(!strName.compare(wstring(ATT_LED_BATF)))
                  {
                        // get the attribute value
                        strValue = saValues(i);

                        // if the led is off...
                        if(!strValue.compare(wstring(VAL_LED_OFF)))
                              m_chkBATF.SetCheck(0);
                        // if it is not off it is on (we don't show blinking
LEDs)
                        else
                              m_chkBATF.SetCheck(1);
                  }
                  // if it's the intf led attribute...
                  else if(!strName.compare(wstring(ATT_LED_INTF)))
                  {
                        // get the attribute value
                        strValue = saValues(i);

                        // if the led is off...
                        if(!strValue.compare(wstring(VAL_LED_OFF)))
                              m_chkINTF.SetCheck(0);
                        // if it is not off it is on (we don't show blinking
LEDs)
                        else
                              m_chkINTF.SetCheck(1);
                  }
                  // if it's the extf led attribute...
                  else if(!strName.compare(wstring(ATT_LED_EXTF)))
                  {
                        // get the attribute value
                        strValue = saValues(i);

                        // if the led is off...
                        if(!strValue.compare(wstring(VAL_LED_OFF)))
                              m_chkEXTF.SetCheck(0);
                        // if it is not off it is on (we don't show blinking
LEDs)
                        else
                              m_chkEXTF.SetCheck(1);
                  }


                  // if it's the busf1 led attribute...
                  else if(!strName.compare(wstring(ATT_LED_BUSF + (TCHAR)'0')))
                  {
                        // get the attribute value
                        strValue = saValues(i);

                        // if the led is off...
                        if(!strValue.compare(wstring(VAL_LED_OFF)))
                              m_chkBUSF1.SetCheck(0);
                        // if it is not off it is on (we don't show blinking
LEDs)
                        else
                              m_chkBUSF1.SetCheck(1);
                  }
                  // if it's the busf2 led attribute...
                  else if(!strName.compare(wstring(ATT_LED_BUSF + (TCHAR)'1')))
                  {
                        // get the attribute value
                        strValue = saValues(i);

                        // if the led is off...
                        if(!strValue.compare(wstring(VAL_LED_OFF)))
                              m_chkBUSF2.SetCheck(0);
                        // if it is not off it is on (we don't show blinking
LEDs)
                        else
                              m_chkBUSF2.SetCheck(1);
                  }
                  // if it's the frce led attribute...
                  else if(!strName.compare(wstring(ATT_LED_FORCE)))
                  {
                        // get the attribute value
                        strValue = saValues(i);

                        // if the led is off...
                        if(!strValue.compare(wstring(VAL_LED_OFF)))
                              m_chkFRCE.SetCheck(0);
                        // if it is not off it is on (we don't show blinking
LEDs)
                        else
                              m_chkFRCE.SetCheck(1);
                  }
```

```
                        // if it's the run led attribute...
                        else if(!strName.compare(wstring(ATT_LED_RUN)))
                        {
                              // get the attribute value
                              strValue = saValues(i);

                              // if the led is off...
                              if(!strValue.compare(wstring(VAL_LED_OFF)))
                                    m_chkRUN.SetCheck(0);
                              // if it is not off it is on (we don't show blinking
LEDs)
                              else
                                    m_chkRUN.SetCheck(1);
                        }
                        // if it's the stop led attribute...
                        else if(!strName.compare(wstring(ATT_LED_STOP)))
                        {
                              // get the attribute value
                              strValue = saValues(i);

                              // if the led is off...
                              if(!strValue.compare(wstring(VAL_LED_OFF)))
                                    m_chkSTOP.SetCheck(0);
                              // if it is not off it is on (we don't show blinking
LEDs)
                              else
                                    m_chkSTOP.SetCheck(1);
                        }
                  }
            }
            // if it's the plc feature...
            else if(!strFeatureName.compare(wstring(FEATURE_PLC)))
            {
                  // go thru all attributes...
                  for(UINT i=0; i<saNames.GetSize(); i++)
                  {
                        // get the attribute name
                        strName = saNames(i);

                        // if it's the plc attribute...
                        if(!strName.compare(wstring(ATT_PLC)))
                        {
                              // get the attribute value
                              strValue = saValues(i);

                              // if the PLC is created...
                              if(!strValue.compare(wstring(VAL_PLC_CREATED)))
                                    ActivateControls();
                              // if the PLC not running...
                              else
                                    DeactivateControls();
                        }
                  }
            }
            // if it's the error feature...
            else if(!strFeatureName.compare(wstring(FEATURE_ERROR)))
            {
```

```
            // go thru all attributes...
            for(UINT i=0; i<saNames.GetSize(); i++)
            {
                  // get the attribute name
                  strName = saNames(i);

                  // if it's the error id attribute...
                  if(!strName.compare(wstring(ATT_ERROR_ID)))
                  {
                        // get the attribute value
                        strValue = saValues(i);

                        // if it's an error....
                        if(_wtol(strValue.c_str()) != _PSERR_OKAY)
                        {
                              // create message
                              string strMessage;
                              strMessage = _T("The FeatureProvider returned an
error.");

                              // show error message
                              MessageBox(strMessage.c_str());
                       }
                  }
           }
      }

      // detach the arrays
      AttributeNames = saNames.Detach();
      AttributeValues = saValues.Detach();

      return S_OK;
}   // end of OnFeatureChanged

STDMETHODIMP CDemoDlg::OnPLCDisconnect(long ErrorID)
{
      // set the application as disconnected
      ConnectionLost();

      return S_OK;
}

HRESULT CDemoDlg::Connect()
{
      // init
      wstring strInstanceName;
      CSafeVariantArray saAttributeNames;
      HRESULT hr;

      // get the instance name
      m_edtInstanceName.GetWindowText(strInstanceName);

      // disable the instance name box
      m_edtInstanceName.EnableWindow(FALSE);

      // connect to the PLC
      hr = this->ConnectToPlc(strInstanceName, &m_pIFeature, m_saFeatureNames,
saAttributeNames);

	// if connection succeeded...
	if(SUCCEEDED(hr))
	{
		// register all necessary features
		hr = this->RegisterAllFeatures();
	}

	return hr;
}

void CDemoDlg::Disconnect()
{
	// unregister all features
	this->UnregisterAllFeatures();

	// clear the plc proxy
	this->ClearPlcData();

	// deactivate all controls
	this->DeactivateControls();

	// clear the plc type info
	m_edtType.SetWindowText(_T(""));

	// enable the instance name box
	m_edtInstanceName.EnableWindow();


}

void CDemoDlg::ConnectionLost()
{
	// clear the plc proxy
	this->ClearPlcData();

	// deactivate all controls
	this->DeactivateControls();

	// change the connect button
	m_btnConnect.SetWindowText(BTN_TXT_CONNECT);

	// disable the connect button => app needs to be restarted
	m_btnConnect.EnableWindow(FALSE);


	// clear the plc type info
	m_edtType.SetWindowText(_T(""));
}

void CDemoDlg::DeactivateControls()
{
	// uncheck all radio buttons
	m_chkON.SetCheck(0);
	m_chkBATF.SetCheck(0);
	m_chkINTF.SetCheck(0);
	m_chkEXTF.SetCheck(0);
	m_chkBUSF1.SetCheck(0);
	m_chkBUSF2.SetCheck(0);
	m_chkFRCE.SetCheck(0);
	m_chkRUN.SetCheck(0);
	m_chkSTOP.SetCheck(0);
	m_chkRUNPbtn.SetCheck(0);
	m_chkRUNbtn.SetCheck(0);
	m_chkSTOPbtn.SetCheck(0);

	// deactivate buttons
	m_btnRUNP.EnableWindow(FALSE);
	m_btnRUN.EnableWindow(FALSE);
	m_btnSTOP.EnableWindow(FALSE);
	m_btnMRES.EnableWindow(FALSE);
	m_btnType.EnableWindow(FALSE);
}


void CDemoDlg::ActivateControls()
{
	// activate buttons
	m_btnRUNP.EnableWindow();
	m_btnRUN.EnableWindow();
	m_btnSTOP.EnableWindow();
	m_btnMRES.EnableWindow();
	m_btnType.EnableWindow();
}

HRESULT CDemoDlg::Browse(CSafeStringArray& saConnectStrings, CSafeStringArray&
saStartInfos)
{
	// clear the variants
	saConnectStrings.Clear();
	saStartInfos.Clear();

	// init
	HRESULT hr;
	long ErrorID = 0;

	// browse for configured and/or running PLCs
	hr = m_pIPlc->Browse(saConnectStrings, saStartInfos, &ErrorID);

	// error?
	if(FAILED(hr))return hr;
	if(ErrorID)return E_FAIL;

	return S_OK;
}

HRESULT CDemoDlg::ConnectToPlc(wstring InstanceName, IFeature** ppIFeature,
CSafeStringArray& saFeatureNames, CSafeVariantArray& saAttributeNames)
{
	// init
	HRESULT hr;
	CSafeStringArray saConnectStrings;
	CSafeStringArray saStartInfos;
```

```
      wstring strConnectionString;
      WCHAR chrBuffer[256];
      ::ZeroMemory(chrBuffer, 256 * sizeof(WCHAR));

      // get connection strings for all available PLCs
      hr = this->Browse(saConnectStrings, saStartInfos);
      if(FAILED(hr))return hr;

      // go thru all returned connection strings and
      // look for the requested instance name
      for(UINT i=0; i<saConnectStrings.GetSize(); i++)
      {
            // get connection string
            strConnectionString = saConnectStrings(i);

            // extract the instance name out of the connection string
            int pos = strConnectionString.find(L"\\", 0);
            int len = strConnectionString.find(L"\\", pos + 1) - pos - 1;
            strConnectionString.copy(chrBuffer, len, pos + 1);
            strConnectionString = chrBuffer;

            // if we have found the correct connection string...
            if(!strConnectionString.compare(InstanceName))
            {
                  // get connection string
                  strConnectionString = saConnectStrings(i);

                  // init
                  long ErrorID;
                  BSTR bstrConnectionString =
::SysAllocString(strConnectionString.c_str());
                  *ppIFeature = NULL;
                  saFeatureNames.Clear();
                  saAttributeNames.Clear();

                  // connect to the PLC
                  hr = m_pIPlc->Connect(bstrConnectionString, ppIFeature,
saFeatureNames, saAttributeNames, &ErrorID);

                  // clean up
                  ::SysFreeString(bstrConnectionString);

                  // error?
                  if(FAILED(hr))return hr;
                  if(ErrorID)return E_FAIL;

                  // we're done
                  return S_OK;
            }
      }

      // we couldn't connect
      return E_FAIL;
}


void CDemoDlg::ClearPlcData()
```

```
{
      m_ConnectionNotifyID = -1;
      m_KeyswitchNotifyID  = -1;
      m_LedNotifyID        = -1;
      m_PlcNotifyID        = -1;
      m_ErrorNotifyID      = -1;

      // release the controller management interface
      if(m_pIFeature.GetInterfacePtr())m_pIFeature.Release();
}

HRESULT CDemoDlg::RegisterAllFeatures()
{
      // error
      if(!m_pIFeature.GetInterfacePtr())return E_FAIL;

      // init
      long ErrorID;
      wstring strFeatureName;
      HRESULT hr;

      // register to get a notification when the connection to the PLC is lost
      hr = m_pIFeature->RegisterForConnectionCheck(this, &m_ConnectionNotifyID,
&ErrorID);

      // error?
      if(FAILED(hr))return hr;
      if(ErrorID)return E_FAIL;

     // go thru all features.....
       for(UINT i=0; i<m_saFeatureNames.GetSize(); i++)
       {
         // get the feature name
         strFeatureName = m_saFeatureNames(i);

         // if it's one of the features we need...
      if(!strFeatureName.compare(wstring(FEATURE_KEYSWITCH)))
      {
            // ...register it
            hr = this->RegisterFeature(strFeatureName, m_KeyswitchNotifyID);
      }
      else if(!strFeatureName.compare(wstring(FEATURE_LED)))
      {
            // ...register it
            hr = this->RegisterFeature(strFeatureName, m_LedNotifyID);
      }
      else if(!strFeatureName.compare(wstring(FEATURE_PLC)))
      {
            // ...register it
            hr = this->RegisterFeature(strFeatureName, m_PlcNotifyID);
      }
      else if(!strFeatureName.compare(wstring(FEATURE_ERROR)))
      {
            // ...register it
            hr = this->RegisterFeature(strFeatureName, m_ErrorNotifyID);
      }
      // error?
      if(FAILED(hr))return hr;
 }

 return S_OK;
}


void CDemoDlg::UnregisterAllFeatures()
{
 // error
 if(!m_pIFeature.GetInterfacePtr())return;

 // let's unregister the features => the feature provider stopps sending
notifications
 if(m_KeyswitchNotifyID != -1)
      this->UnregisterFeature(wstring(FEATURE_KEYSWITCH), m_KeyswitchNotifyID);
 if(m_LedNotifyID != -1)
      this->UnregisterFeature(wstring(FEATURE_LED), m_LedNotifyID);
 if(m_PlcNotifyID != -1)
      this->UnregisterFeature(wstring(FEATURE_PLC), m_PlcNotifyID);
 if(m_ErrorNotifyID != -1)
      this->UnregisterFeature(wstring(FEATURE_ERROR), m_ErrorNotifyID);

     // unregister for connection check
 if(m_ConnectionNotifyID != -1)
 {
      long ErrorID;
      m_pIFeature->UnregisterForConnectionCheck(m_ConnectionNotifyID,
&ErrorID);
 }
}
HRESULT CDemoDlg::RegisterFeature(wstring FeatureName, long& NotificationID)
{
 // error
 if(!m_pIFeature.GetInterfacePtr())return E_FAIL;

 // init
  HRESULT hr;
 long ErrorID;
 _variant_t var;
 BSTR bstrFeatureName;

 // create a BSTR
 bstrFeatureName = ::SysAllocString(FeatureName.c_str());

 // create a context
 var.ulVal = (DWORD)this;

    // register the feature to get change notifications
  hr = m_pIFeature->RegisterFeatureForChange(this, bstrFeatureName, var,
&NotificationID, &ErrorID);


     // clean up
 ::SysFreeString(bstrFeatureName);
 // error?
  if(FAILED(hr))return hr;
 if(ErrorID)return E_FAIL;

 return S_OK;
}
HRESULT CDemoDlg::UnregisterFeature(wstring FeatureName, long NotificationID)
{
 // error
 if(!m_pIfeature.GetInterfacePtr())return E_FAIL;

 // init
 HRESULT hr;
 long ErrorID;

    // unregister the feature to cancel notifications
  hr = m_pIFeature->UnregisterFeatureForChange(NotificationID, &ErrorID);

 // error?
 if(FAILED(hr))return hr;
 if(ErrorID)return E_FAIL;

 return S_OK;
}

HRESULT CDemoDlg::SetKeyswitch(wstring Value)
{
 // error
 if(!m_pIFeature.GetInterfacePtr())return E_FAIL;

 // init
 BSTR bstrFeatureName;
 long ErrorID;
  HRESULT hr;
 CSafeStringArray saAttributeNames;
 CSafeStringArray saAttributeValues;

 // create arrays
  saAttributeNames.SetSize(1);
 saAttributeValues.SetSize(1);

 // set values
  saAttributeNames.PutElement(0,
::SysAllocString(wstring(ATT_KEYSWITCH).c_str()));
 saAttributeValues.PutElement(0, ::SysAllocString(Value.c_str()));

 // create a BSTR
  bstrFeatureName = ::SysAllocString(wstring(FEATURE_KEYSWITCH).c_str());

 // set the keyswitch
 hr = m_pIfeature->SetFeature(bstrFeatureName, saAttributeNames,
saAttributeValues, &ErrorID);

 // clean up
  ::SysFreeString(bstrFeatureName);

 // error?
 if(FAILED(hr))return hr;
 if(ErrorID)return E_FAIL;

 return S_OK;
}

HRESULT CDemoDlg::GetCpuType(wstring& Value)
{
 // error
 if(!m_pIFeature.GetInterfacePtr())return E_FAIL;

 // init
 Value = L"";
 BSTR bstrFeatureName;
 long ErrorID;
 HRESULT hr;
 CSafeStringArray saAttributeNames;
 CSafeStringArray saAttributeValues;

 // create a BSTR
  bstrFeatureName = ::SysAllocString(wstring(FEATURE_PERSONALITY).c_str());

     // get the personality feature attribute values
 hr = m_pIFeature->GetFeature(bstrFeatureName, saAttributeNames,
saAttributeValues, &ErrorID);

 // clean up
 ::SysFreeString(bstrFeatureName);

 // error?
 if(FAILED(hr))return hr;
 if(ErrorID)return E_FAIL;

     // loop thru all attributes
 for(UINT i=0; i<saAttributeNames.GetSize(); i++)
 {
      // if it's the plc type attribute
      if(!wstring(saAttributeNames(i)).compare(wstring(ATT_PERSONALITY_TYPE)))
      {
            // get type
            Value = saAttributeValues(i);

            return S_OK;
      }
 }

 return S_FALSE;
}

LRESULT CDemoDlg::OnConnectBtn(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled)
{
 // lock the button
 m_btnConnect.EnableWindow(FALSE);

 // init
 string strBtnText;
 // get the button text
 m_btnConnect.GetWindowText(strBtnText);

 // if we want to connect...
 if(!strBtnText.compare(BTN_TXT_CONNECT))
 {
      // init
      HRESULT hr;

      // connect to the PLC
      hr = this->Connect();

      // if everything is fine...
      if(SUCCEEDED(hr))
      {
            // change the connect button
            m_btnConnect.SetWindowText(BTN_TXT_DISCONNECT);
      }
      // if connection failed...
      else
      {
            // disconnect the PLC and clear all settings
            this->Disconnect();

            // show error message box
            MessageBox(_T("Connecting to the PLC failed."));
      }
 }
 // if we want to disconnect...
 else if(!strBtnText.compare(BTN_TXT_DISCONNECT))
 {
      // disconnect the PLC
      this->Disconnect();

      // change the connect button
      m_btnConnect.SetWindowText(BTN_TXT_CONNECT);
 }

 // unlock the button
 m_btnConnect.EnableWindow();

 return 0;
}
LRESULT CDemoDlg::OnMresBtn(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled)
{
     // create message
     string strMessage = _T("The module will be reset (clear/reset).  All user
data will be deleted and all existing connections to the module will be
disconnected.  Do you really want to reset the module?");

     // double-check with the user
     if(::MessageBox(NULL, strMessage.c_str(), _T("Demo"),
MB_YESNO|MB_ICONWARNING|MB_APPLMODAL) == IDYES)

          // send a request to change the keyswitch to MRES
          this->SetKeyswitch(wstring(VAL_KEYSWITCH_MRES));
     }
     return 0;
}


LRESULT CDemoDlg::OnRunBtn(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled)
{
 // send a request to change the keyswitch to RUN
 this->SetKeyswitch(wstring(VAL_KEYSWITCH_RUN));

 return 0;
}
LRESULT CDemoDlg::OnRunpBtn(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled)
{
 // send a request to change the keyswitch to RUN-P
 this->SetKeyswitch(wstring(VAL_KEYSWITCH_RUNP));

 return 0;
}
LRESULT CDemoDlg::OnStopBtn(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled)
{
 // send a request to change the keyswitch to STOP
 this->SetKeyswitch(wstring(VAL_KEYSWITCH_STOP));

 return 0;
}

LRESULT CDemoDlg::OnGetType(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled)
{
 // init
 HRESULT hr;
 wstring strPlcType;

 // get the plc type info
 hr = this->GetCpuType(strPlcType);

 // error
 if(FAILED(hr))
      strPlcType = L"error";
 else if(hr == S_FALSE)
      strPlcType = L"unknown";

 // display the plc type
  m_edtType.SetWindowText(strPlcType);

 return 0;
}
```

Header Files

DemoDlg.h

```
// DemoDlg.h : Declaration of the CDemoDlg

#ifndef __DEMODLG_H_
#define __DEMODLG_H_

#include "resource.h"       // main symbols
#include <atlhost.h>
#include "Demo.h"
#include "Definitions.h"

_COM_SMARTPTR_TYPEDEF(IFeatureCallback, __uuidof(IFeatureCallback));

#define BTN_TXT_CONNECT "Connect"
#define BTN_TXT_DISCONNECT "Disconnect"

/////////////////////////////////////////////////////////////////////////////
// CDemoDlg
class CDemoDlg :
      public CAxDialogImpl<CDemoDlg>,
      public CComObjectRootEx<CComSingleThreadModel>,
      public CComCoClass<CDemoDlg, &CLSID_DemoDlg>,
      public IDispatchImpl<IFeatureCallback, &IID_IFeatureCallback,
&LIBID_DEMOLib>
{
public:
      CDemoDlg()
      {
            // init
            this->ClearPlcData();

            // create an instance of the feature provider
            m_pIPlc.CreateInstance(__uuidof(PLC));
      }

      ~CDemoDlg()
      {
            // release feature provider
            m_pIPlc.Release();
      }

      enum { IDD = IDD_DEMODLG };

DECLARE_REGISTRY_RESOURCEID(IDR_DEMO)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CDemoDlg)
      COM_INTERFACE_ENTRY(IFeatureCallback)
      COM_INTERFACE_ENTRY(IDispatch)
END_COM_MAP()

BEGIN_MSG_MAP(CDemoDlg)
```

```
	MESSAGE_HANDLER(WM_INITDIALOG, OnInitDialog)
	COMMAND_ID_HANDLER(IDCANCEL, OnCancel)
	COMMAND_HANDLER(IDC_CONNECT_BTN, BN_CLICKED, OnConnectBtn)
	COMMAND_HANDLER(IDC_MRES_BTN, BN_CLICKED, OnMresBtn)
	COMMAND_HANDLER(IDC_RUN_BTN, BN_CLICKED, OnRunBtn)
	COMMAND_HANDLER(IDC_RUNP_BTN, BN_CLICKED, OnRunpBtn)
	COMMAND_HANDLER(IDC_STOP_BTN, BN_CLICKED, OnStopBtn)
	COMMAND_HANDLER(IDC_GET_TYPE, BN_CLICKED, OnGetType)
END_MSG_MAP()

	// IFeatureCallback interface methods
	STDMETHOD(OnFeatureChanged)(BSTR FeatureName, VARIANT Context, long
NotificationID, VARIANT AttributeNames, VARIANT AttributeValues);
	STDMETHOD(OnPLCDisconnect)(long ErrorID);

	// dialog management
	LRESULT OnInitDialog(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL&
bHandled);

	// message handler
	LRESULT OnCancel(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled);
	LRESULT OnConnectBtn(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled);
	LRESULT OnMresBtn(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled);
	LRESULT OnRunBtn(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled);
	LRESULT OnRunpBtn(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled);
	LRESULT OnStopBtn(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled);
	LRESULT OnGetType(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL&
bHandled);

	// panel methods
	void	ActivateControls();
	void	DeactivateControls();
	HRESULT Connect();
	void	Disconnect();
	void	ConnectionLost();
	HRESULT Browse(CSafeStringArray& saConnectStrings, CSafeStringArray&
saStartInfos);
	HRESULT ConnectToPlc(wstring InstanceName, IFeature** ppIFeature,
CSafeStringArray& saFeatureNames, CSafeVariantArray& saAttributeNames);
	void	ClearPlcData();
	HRESULT RegisterAllFeatures();
	void	UnregisterAllFeatures();
	HRESULT RegisterFeature(wstring FeatureName, long& NotificationID);
	HRESULT UnregisterFeature(wstring FeatureName, long NotificationID);
	HRESULT SetKeyswitch(wstring Value);
	HRESULT GetCpuType(wstring& Value);

private:
	// all dialog controls
	CMyCheckBox m_chkON;
	CMyCheckBox m_chkBATF;
```

```
      CMyCheckBox m_chkINTF;
      CMyCheckBox m_chkEXTF;
      CMyCheckBox m_chkBUSF1;
      CMyCheckBox m_chkBUSF2;
      CMyCheckBox m_chkFRCE;
      CMyCheckBox m_chkRUN;
      CMyCheckBox m_chkSTOP;
      CMyCheckBox m_chkRUNPbtn;
      CMyCheckBox m_chkRUNbtn;
      CMyCheckBox m_chkSTOPbtn;
      CMyButton      m_btnConnect;
      CMyButton      m_btnType;
      CMyButton      m_btnRUNP;
      CMyButton      m_btnRUN;
      CMyButton      m_btnSTOP;
      CMyButton      m_btnMRES;
      CMyEditBox      m_edtInstanceName;
      CMyEditBox      m_edtType;

      // notification IDs
      long m_ConnectionNotifyID;
      long m_KeyswitchNotifyID;
      long m_LedNotifyID;
      long m_PlcNotifyID;
      long m_ErrorNotifyID;

      // interface pointer
      IPLCPtr           m_pIPlc;
      IFeaturePtr m_pIFeature;

      // available features
      CSafeStringArray m_saFeatureNames;
};

#endif //__DEMODLG_H_
```

StdAfx.h

```
// stdafx.h : include file for standard system include files,
//      or project specific include files that are used frequently,
//      but are changed infrequently

#if !defined(AFX_STDAFX_H__48133E64_2594_49FA_8664_1C7953A67A8B__INCLUDED_)
#define AFX_STDAFX_H__48133E64_2594_49FA_8664_1C7953A67A8B__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define STRICT
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0400
#endif
#define _ATL_APARTMENT_THREADED

#include <atlbase.h>
//You may derive a class from CComModule and use it if you want to override
//something, but do not change the name of _Module
class CExeModule : public CComModule
{
public:
      LONG Unlock();
      DWORD dwThreadID;
      HANDLE hEventShutdown;
      void MonitorShutdown();
      bool StartMonitor();
      bool bActivity;
};
extern CExeModule _Module;
#include <atlcom.h>
#include <atlwin.h>

#include <string>
#include "ComDef.h"
#include "SafeArray.h"
#include "strDef.h"
#include "featureStrDefine.h"
using namespace std;

#import  "FeatureProvider.tlb" no_namespace
#import  "Interfaces.tlb" no_namespace exclude("IFeatureCallback")

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_STDAFX_H__48133E64_2594_49FA_8664_1C7953A67A8B__INCLUDED)
```

External Dependencies

featureStrDefine.h

```
/***************************************************************************
*   Copyright                *
***************************************************************************

*--------------------------------------------------------------------------
*
*   Author       :   Max Wang
*   Date         :   03/04/02
*
*--------------------------------------------------------------------------
*
*   Description :   feature/attribute/value string defines
*
*--------------------------------------------------------------------------
*
*   Modification history:
*
\**************************************************************************/
#ifndef __FEATURESTRDEFINE_H
#define __FEATURESTRDEFINE_H

#ifdef __cplusplus
extern "C" {
#endif

'*--------------------------------------------------------------------
      LED
--------------------------------------------------------------------*/
#define FEATURE_LED                           _C("LED")

/* attribute power LED */
#define ATT_LED_POWER                         _C("Power")

/* attribute battery fault LED */
#define ATT_LED_BATF                          _C("BatteryFault")

/* attribute internal fault LED */
#define ATT_LED_INTF                          _C("InternalFault")

/* attribute external fault LED */
#define ATT_LED_EXTF                          _C("ExternalFault")

/* attribute bus fault LED */
#define ATT_LED_BUSFAULTCOUNT                 _C("BusFaultCount")
/*        VAL_LED_BUSFAULTCOUNT               %d */
#define ATT_LED_BUSF                           _C("BusFault_")

/* attribute force LED */
#define ATT_LED_FORCE                          _C("Force")

/* attribute run LED */
#define ATT_LED_RUN                            _C("Run")
```

```
/* attribute stop LED */
#define ATT_LED_STOP                             _C("Stop")

/* value LED */
#define VAL_LED_ON                               _C("ON")
#define VAL_LED_OFF                              _C("OFF")
#define VAL_LED_BLINKING2HZ                      _C("Blinking2HZ")
#define VAL_LED_BLINKING05HZ                     _C("Blinking05HZ")

'*----------------------------------------------------------------------
      KeySwitch
-----------------------------------------------------------------------*/
#define FEATURE_KEYSWITCH                        _C("KeySwitch")

/* attribute KeySwitch */
#define ATT_KEYSWITCH                            _C("Value")
#define VAL_KEYSWITCH_MRES                       _C("MRES")
#define VAL_KEYSWITCH_STOP                       _C("STOP")
#define VAL_KEYSWITCH_RUN                        _C("RUN")
#define VAL_KEYSWITCH_RUNP                       _C("RUNP")

/* attribute force coldstart */
#define ATT_KEYSWITCH_FORCECOLDSTART             _C("ForceColdstart")
/*      #define VAL_ON                           _C("On")
        #define VAL_OFF                          _C("Off") */

'*----------------------------------------------------------------------
      PLCInstance
-----------------------------------------------------------------------*/
#define FEATURE_PLCINSTANCE                      _C("PLCInstance")

/* attribute PLCINSTANCE */
#define ATT_PLCINSTANCE                          _C("Value")
#define VAL_PLCINSTANCE_CREATE                   _C("Create")
#define      VAL_PLCINSTANCE_SHUTDOWN            _C("Shutdown")

'*----------------------------------------------------------------------
      PLCPower
-----------------------------------------------------------------------*/
#define FEATURE_PLCPOWER                         _C("PLCPower")

/* attribute PLCPower state */
#define ATT_PLCPOWER                             _C("Value")
#define VAL_ON                                   _C("On")
#define VAL_OFF                                  _C("Off")

'*----------------------------------------------------------------------
      PLC
-----------------------------------------------------------------------*/
#define FEATURE_PLC                              _C("PLC")

/* attribute PLC */
#define ATT_PLC                                  _C("Value")
#define VAL_PLC_CREATED                          _C("Created")
#define VAL_PLC_SHUTDOWNED                       _C("Shutdowned")
#define VAL_PLC_NOTAVAILABLE                     _C("NotAvailable")
```

```
'*---------------------------------------------------------------------
      StartAtBoot
----------------------------------------------------------------------*/
#define FEATURE_STARTATBOOT                     _C("StartAtBoot")

/* attribute Start at boot */
#define ATT_STARTATBOOT                         _C("Value")
/*      #define VAL_ON                          _C("On")
        #define VAL_OFF                         _C("Off") */

'*---------------------------------------------------------------------
      AutoStart
----------------------------------------------------------------------*/
#define FEATURE_AUTOSTART                       _C("AutoStart")

/* attribute Autostart */
#define ATT_AUTOSTART                           _C("Value")
/*      #define VAL_ON                          _C("On")
        #define VAL_OFF                         _C("Off") */

'*---------------------------------------------------------------------
      AutoLoad
----------------------------------------------------------------------*/
#define FEATURE_AUTOLOAD                        _C("AutoLoad")

/* attribute Autoload enabled */
#define ATT_AUTOLOAD                            _C("Value")
/*      #define VAL_ON                          _C("On")
        #define VAL_OFF                         _C("Off") */

/* attribute Keyswitch state after Autoload */
#define ATT_AUTOLOAD_KEYSWITCH                  _C("KeySwitch")
#define VAL_AUTOLOAD_KS_STOP                    _C("STOP")
#define VAL_AUTOLOAD_KS_RUN                     _C("RUN")
#define VAL_AUTOLOAD_KS_RUNP                    _C("RUNP")

/* attribute Buffer for Autoload */
#define ATT_AUTOLOAD_BUFFER                     _C("Buffer")
/*      #define VAL_AUTOLOAD_BUFFER                         %s */

/* attribute Buffersize for Autoload */
#define ATT_AUTOLOAD_BUFFERSIZE                 _C("BufferSize")
/*      #define VAL_AUTOLOAD_BUFFERSIZE              %d */

/* attribute target filename for Autoload */
#define ATT_AUTOLOAD_TARGETFILE                 _C("TargetFile")
/*      #define VAL_AUTOLOAD_TARGETFILE              %s */

'*---------------------------------------------------------------------
      Security
----------------------------------------------------------------------*/
#define FEATURE_SECURITY                        _C("Security")

/* attribute actual Password */
#define ATT_SECURITY_ACTPASSWORD                _C("Password")
/*      #define VAL_SECURITY_ACTPASSWORD              %s scrammbled */
/* attribute new Password */
#define ATT_SECURITY_NEWPASSWORD              _C("NewPassword")
/*      #define VAL_SECURITY_NEWPASSWORD            %s scrammbled */

/* attribute Password check */
#define ATT_SECURITY_PASSWORDCHECK            _C("Check")
#define VAL_SECURITY_PASSWORDCHECK_PASS       _C("Passed")
#define VAL_SECURITY_PASSWORDCHECK_FAILED     _C("Failed")

/* attribute security level */
#define ATT_SECURITY_LEVEL                    _C("Level")
#define VAL_SECURITY_LEVELPASSWORD            _C("Password")
#define VAL_SECURITY_LEVELPASSWORDDISABLED    _C("PasswordDisabled")
#define VAL_SECURITY_LEVELCONFIRMATION        _C("Confirmation")
#define VAL_SECURITY_LEVELNONE                _C("None")

/* attribute Password Prompt Interval hours */
#define ATT_SECURITY_INTERVALHOURS            _C("hInterval")
/*      #define VAL_SECURITY_INTERVALHOURS          %d */

/* attribute Password Prompt Interval minutes */
#define ATT_SECURITY_INTERVALMINUTES          _C("mInterval")
/*      #define VAL_SECURITY_INTERVALMINUTES  %d */

'*---------------------------------------------------------------------
      FMR
---------------------------------------------------------------------*/
#define FEATURE_FMR                           _C("FMR")

'*---------------------------------------------------------------------
      MemoryCard file
---------------------------------------------------------------------*/
#define FEATURE_MCF                           _C("MemoryCardFile")

/* attribute MemoryCard file Buffer */
#define ATT_MCF_BUFFER                        _C("Buffer")
/*       VAL_MCF_BUFFER                                        %s */

/* attribute MemoryCard file Size */
#define ATT_MCF_SIZE                          _C("Size")
/*       VAL_MCF_SIZE                                          %d */

'*---------------------------------------------------------------------
      Priority
---------------------------------------------------------------------*/
#define FEATURE_PRIORITY                      _C("Priority")

/* attribute Priority value */
#define ATT_PRIORITY                          _C("Value")
/*      #define VAL_PRIORITY                              %d */

#define ATT_PRIORITY_LOWERLIMIT               _C("LowerLimit")
/*      #define VAL_PRIORITY_LOWERLIMIT            %d */

#define ATT_PRIORITY_UPPERLIMIT               _C("UpperLimit")
/*      #define VAL_PRIORITY_UPPERLIMIT            %d */
```

```
#define ATT_NORMAL_PRIORITY                         _C("Normal")
/*      #define VAL_NORMAL_PRIORITY                             %d */

#define ATT_CRITICAL_PRIORITY                       _C("Critical")
/*      #define VAL_CRITICAL_PRIORITY                           %d */

'*----------------------------------------------------------------------
      Minimum Sleep Time
---------------------------------------------------------------------*/
#define FEATURE_MINSLEEPTIME                        _C("MinSleepTime")

/* attribute Minimum SleepTime value */
#define ATT_MINSLEEPTIME                            _C("Value")
/*      #define VAL_MINSLEEPTIME                          %d */

#define ATT_MINSLEEPTIME_LOWERLIMIT                 _C("LowerLimit")
/*      #define VAL_MINSLEEPTIME_LOWERLIMIT               %d */

#define ATT_MINSLEEPTIME_UPPERLIMIT                 _C("UpperLimit")
/*      #define VAL_MINSLEEPTIME_UPPERLIMIT               %d */

'*----------------------------------------------------------------------
      Minimum Cycle Time
---------------------------------------------------------------------*/
#define FEATURE_MINCYCLETIME                        _C("MinCycleTime")

/* attribute Minimum CycleTime value */
#define ATT_MINCYCLETIME                            _C("Value")
/*      #define VAL_MINCYCLETIME                          %d */

#define ATT_MINCYCLETIME_LOWERLIMIT                 _C("LowerLimit")
/*      #define VAL_MINSLEEPTIME_LOWERLIMIT               %d */

#define ATT_MINCYCLETIME_UPPERLIMIT                 _C("UpperLimit")
/*      #define VAL_MINCYCLETIME_UPPERLIMIT               %d */

'*----------------------------------------------------------------------
      CPU Usage
---------------------------------------------------------------------*/
#define FEATURE_CPUUSAGE                            _C("Usage")

/* attribute Usage count */
#define ATT_PCUSAGE                                 _C("PC")
/*       VAL_PCUSAGE                                               %d [0..100%] */

/* attribute PLC Usage */
#define ATT_PLCUSAGE                                _C("PLC")
/*      #define VAL_PLCUSAGE                                %d [0..100%] */

/* attribute Usage count */
#define ATT_CPUUSAGE_COUNT                          _C("CPUCount")
/*       VAL_CPUUSAGE_COUNT                                        %d */

/* attribute CPU Usage values */
#define ATT_CPUUSAGE                                _C("CPU_")
/*      #define VAL_CPUUSAGE                                %d [0..100%] */
```

```
'*----------------------------------------------------------------------
      Timing
-----------------------------------------------------------------------*/
#define FEATURE_TIMING                                _C("Timing")

/* attribute Timing Upper Limit */
#define ATT_TIMING_UPPERLIMIT                         _C("UpperLimit")
/*        VAL_TIMING_UPPERLIMIT                                  %d */

/* attribute CycleTime count */
#define ATT_TIMING_CYCLETIMECOUNT                     _C("CycleTimeCount")
/*       #define VAL_TIMING_CYCLETIMECOUNT                  %d */

/* attribute CycleTime buffer */
#define ATT_TIMING_CYCLETIMEBUFFER                    _C("CycleTimeBuffer")
/*       #define VAL_TIMING_CYCLETIMEBUFFER                 %d %d... */

/* attribute CycleTime minimum */
#define ATT_TIMING_CYCLETIMEMIN                       _C("CycleTimeMin")
/*       #define VAL_TIMING_CYCLETIMEMIN                   %d */

/* attribute CycleTime maximum */
#define ATT_TIMING_CYCLETIMEMAX                       _C("CycleTimeMax")
/*       #define VAL_TIMING_CYCLETIMEMAX                   %d */

/* attribute CycleTime average */
#define ATT_TIMING_CYCLETIMEAVE                       _C("CycleTimeAverage")
/*       #define VAL_TIMING_CYCLETIMEAVE                   %d */

/* attribute CycleTime last */
#define ATT_TIMING_CYCLETIMELAST                      _C("CycleTimeLast")
/*       #define VAL_TIMING_CYCLETIMELAST                   %d */

/* attribute ExecTime minimum */
#define ATT_TIMING_EXECTIMEMIN                        _C("ExecTimeMin")
/*       #define VAL_TIMING_EXECTIMEMIN                     %d */

/* attribute ExecTime maximum */
#define ATT_TIMING_EXECTIMEMAX                        _C("ExecTimeMax")
/*       #define VAL_TIMING_EXECTIMEMAX                     %d */

/* attribute ExecTime average */
#define ATT_TIMING_EXECTIMEAVE                        _C("ExecTimeAverage")
/*       #define VAL_TIMING_EXECTIMEAVE                     %d */

/* attribute ExecTime last */
#define ATT_TIMING_EXECTIMELAST                       _C("ExecTimeLast")
/*       #define VAL_TIMING_EXECTIMELAST                   %d */

/* attribute SleepIntervalCounter */
#define ATT_TIMING_SLEEPINTERVALCOUNTER               _C("SleepIntervalCounter")
/*        VAL_TIMING_SLEEPINTERVALCOUNTER                        %d */

/* attribute Clear */
#define ATT_TIMING_CLEAR                              _C("Clear")
/*       #define VAL_TIMING_CLEAR                           empty */
```

```
'*-----------------------------------------------------------------------
      OBExecution
-----------------------------------------------------------------------*/
#define FEATURE_OBEXEC                        _C("OBExecution")

/* attribute WakeInterval */
#define ATT_OBEXEC_WAKEINTERVAL               _C("WakeInterval")
/*       VAL_OBEXEC_WAKEINTERVAL                         %d */

/* attribute SleepInterval */
#define ATT_OBEXEC_SLEEPINTERVAL              _C("SleepInterval")
/*       VAL_OBEXEC_SLEEPINTERVAL                        %d */

/* attribute default WakeInterval */
#define ATT_OBEXEC_DEFAULTWAKEINTERVAL        _C("DefaultWakeInterval")
/*       VAL_OBEXEC_DEFAULTWAKEINTERVAL                  %d */

/* attribute default SleepInterval */
#define ATT_OBEXEC_DEFAULTSLEEPINTERVAL       _C("DefaultSleepInterval")
/*       VAL_OBEXEC_DEFAULTSLEEPINTERVAL                 %d */

/* attribute UpperLimit maximum execution load */
#define ATT_OBEXEC_UPPERLIMIT                 _C("UpperLimit")
/*       VAL_OBEXEC_UPPERLIMIT                           %d [0..100%] */

/* attribute Lowerimit maximum execution load */
#define ATT_OBEXEC_LOWERLIMIT                 _C("LowerLimit")
/*       VAL_OBEXEC_LOWERLIMIT                           %d [0..100%] */

'*-----------------------------------------------------------------------
      Diagnostic Language
-----------------------------------------------------------------------*/
#define FEATURE_DIAGLANGUAGE                  _C("DiagnosticLanguage")

/* attribute Language count */
#define ATT_DIAGLANGUAGE_COUNT               _C("Count")
/*       VAL_DIAGLANGUAGE_COUNT                          %d

/* attribute Language */
#define ATT_DIAGLANGUAGE                     _C("Language_")
#define      VAL_LANGUAGE_GERMAN             _C("GERMAN")
#define      VAL_LANGUAGE_ENGLISH            _C("ENGLISH")
#define      VAL_LANGUAGE_FRENCH             _C("FRENCH")
#define      VAL_LANGUAGE_ITALIAN            _C("ITALIAN")
#define      VAL_LANGUAGE_SPANISH            _C("SPANISH")
#define      VAL_LANGUAGE_JAPANESE           _C("JAPANESE")
#define      VAL_LANGUAGE_CHINESE            _C("CHINESE")

'*-----------------------------------------------------------------------
      Diagnostic Information
-----------------------------------------------------------------------*/
#define FEATURE_DIAGNOSTIC                   _C("Diagnostic")

/* attribute Language */
#define ATT_DIAG_LANGUAGE                    _C("Language")
/*       VAL_DIAG_LANGUAGE                   %s */
/* attribute Diagnostic entry count */
#define ATT_DIAG_COUNT                           _C("Count")
/*       VAL_DIAG_COUNT                          %d

/* attribute Time */
#define ATT_DIAG_TIME                            _C("Time_")
/*       VAL_DIAG_TIME                           %s */

/* attribute Date */
#define ATT_DIAG_DATE                            _C("Date_")
/*       VAL_DIAG_DATE                           %s */

/* attribute Event short text */
#define ATT_DIAG_EVENTSHORT                      _C("EventShort_")
/*       VAL_DIAG_EVENTSHORT                     %s */

/* attribute Event long text */
#define ATT_DIAG_EVENTLONG                       _C("EventLong_")
/*       VAL_DIAG_EVENTLONG                      %s */

/* attribute Event ID */
#define ATT_DIAG_EVENTID                         _C("EventID_")
/*       VAL_DIAG_EVENTID                        %s */

/* attribute Event hex text */
#define ATT_DIAG_EVENTHEX                        _C("EventHex_")
/*       VAL_DIAG_EVENTHEX                       %s */

'*----------------------------------------------------------------------
      Personality
---------------------------------------------------------------------*/
#define FEATURE_PERSONALITY                      _C("Personality")

/* attribute instance name */
#define ATT_PERSONALITY_NAME                     _C("Name")
/*      VAL_PERSONALITY_NAME                     %s */

/* attribute CPU type */
#define ATT_PERSONALITY_TYPE                     _C("Type")
/*      ATT_PERSONALITY_TYPE                      %s */

/* attribute product code type */
#define ATT_PERSONALITY_PRODUCTCODE               _C("ProductCode")
/*      ATT_PERSONALITY_PRODUCTCODE               %s */

/* attribute SW release */
#define ATT_PERSONALITY_SW_RELEASE                _C("SWRelease")
/*      ATT_PERSONALITY_SW_RELEASE                %s */

/* attribute FW release */
#define ATT_PERSONALITY_FW_RELEASE                _C("FWRelease")
/*      ATT_PERSONALITY_FW_RELEASE                %s */

/* attribute HW release */
#define ATT_PERSONALITY_HW_RELEASE                _C("HWRelease")
/*      ATT_PERSONALITY_HW_RELEASE                %s */
/* attribute SLOT Number */
#define ATT_PERSONALITY_SLOT_NUMBER             _C("Slot")
/*      ATT_PERSONALITY_SLOT_NUMBER             %s */

/* attribute RACK Number */
#define ATT_PERSONALITY_RACK_NUMBER             _C("Rack")
/*      ATT_PERSONALITY_RACK_NUMBER             %s */

/* attribute OWNER Info */
#define ATT_PERSONALITY_OWNER_INFO              _C("Owner")
/*      ATT_PERSONALITY_OWNER_INFO              %s */

/* attribute RACK Number */
#define ATT_PERSONALITY_COMPANY_INFO            _C("Company")
/*      ATT_PERSONALITY_COMPANY_INFO            %s */

'*----------------------------------------------------------------------
      Error
----------------------------------------------------------------------*/
#define FEATURE_ERROR                           _C("Error")

/* attribute error identification */
#define ATT_ERROR_ID                            _C("ID")
/*      VAL_ERROR_ID                            %d */

/* valid error identifications */
enum
{
      _PSERR_OKAY                               = 0,
      _PSERR_NO_MEMORY,
      _PSERR_ARCHIVE_NOT_VALID_IN_RUN,
      _PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU,
      _PSERR_RESTORE_CANNOT_LINKIN_BLOCK,
      _PSERR_RESTORE_NOT_VALID_IN_RUN,
      _PSERR_RESTORE_FILE_INVALID,
      _PSERR_INIT_EDBSERVER,
      _PSERR_INIT_PDH,
      _PSERR_KEYSWITCH_NOT_ALLOWED_IN_MCF_OP,
      _PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU_PASSWORD_PROTECTED,
      _PSERR_FILE_SIZE_EXCEEDED,
      _PSERR_FWUPDATE_NOT_SUCCEEDED,
      _PSERR_FWUPDATE_NOT_POSSIBLE
};

'*----------------------------------------------------------------------
      HelpInfo
----------------------------------------------------------------------*/
#define FEATURE_CONTROLLER_HELP                 _C("ControllerHelp")

/* attribute host */
#define ATT_HELP_HOST                           _C("Host")
/*      VAL_HELP_HOST                           %s */

/* attribute helpsystem */
#define ATT_HELP_SYSTEM                         _C("HelpSystem")
#define VAL_HELP_SYSTEM_WIN                     _C("WinHelp")
```

```
#define VAL_HELP_SYSTEM_WEB                     _C("WebHelp")
#define VAL_HELP_SYSTEM_HTML                    _C("HTMLHelp")

/* attribute document help directory; includes drive letter */
#define ATT_HELP_DIR                            _C("HelpDir")
/*      VAL_HELP_DIR                            %s */

/* attribute Language count */
#define ATT_HELPLANGUAGE_COUNT                   _C("Count")
/*       VAL_HELPLANGUAGE_COUNT                  %d

/* attribute Language */
#define ATT_HELPLANGUAGE                         _C("Language_")
/*
#define      VAL_LANGUAGE_GERMAN                 _C("GERMAN")
#define      VAL_LANGUAGE_ENGLISH                _C("ENGLISH")
#define      VAL_LANGUAGE_FRENCH                 _C("FRENCH")
#define      VAL_LANGUAGE_ITALIAN                _C("ITALIAN")
#define      VAL_LANGUAGE_SPANISH                _C("SPANISH")
#define      VAL_LANGUAGE_JAPANESE               _C("JAPANESE")
#define      VAL_LANGUAGE_CHINESE                _C("CHINESE")
*/

'*----------------------------------------------------------------------
      CPU Language
---------------------------------------------------------------------*/
#define FEATURE_CPULANGUAGE                      _C("CPULanguage")

/* attribute Language */
#define ATT_CPULANGUAGE_CURRENT                  _C("CurrentLanguage")
/*
#define      VAL_LANGUAGE_GERMAN                 _C("GERMAN")
#define      VAL_LANGUAGE_ENGLISH                _C("ENGLISH")
#define      VAL_LANGUAGE_FRENCH                 _C("FRENCH")
#define      VAL_LANGUAGE_ITALIAN                _C("ITALIAN")
#define      VAL_LANGUAGE_SPANISH                _C("SPANISH")
#define      VAL_LANGUAGE_JAPANESE               _C("JAPANESE")
#define      VAL_LANGUAGE_CHINESE                _C("CHINESE")
*/

/* attribute Language count */
#define ATT_CPULANGUAGE_COUNT                    _C("Count")
/*       VAL_CPULANGUAGE_COUNT                   %d

/* attribute Language */
#define ATT_CPULANGUAGE                          _C("Language_")
/*
#define      VAL_LANGUAGE_GERMAN                 _C("GERMAN")
#define      VAL_LANGUAGE_ENGLISH                _C("ENGLISH")
#define      VAL_LANGUAGE_FRENCH                 _C("FRENCH")
#define      VAL_LANGUAGE_ITALIAN                _C("ITALIAN")
#define      VAL_LANGUAGE_SPANISH                _C("SPANISH")
#define      VAL_LANGUAGE_JAPANESE               _C("JAPANESE")
#define      VAL_LANGUAGE_CHINESE                _C("CHINESE")
*/

/*----------------------------------------------------------------------
```

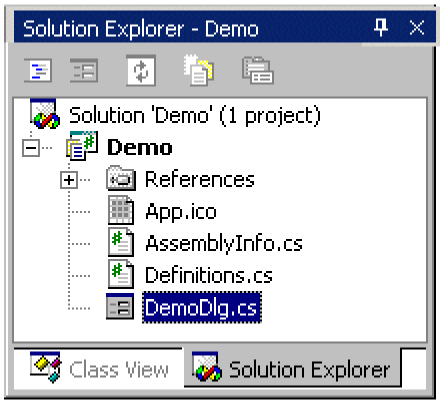
```
      PC/PG Interface
----------------------------------------------------------------------*/
#define FEATURE_PCPGINTERFACE                          _C("PC_PG_Interface")

/*---------------------------------------------------------------------
Firmware Update
----------------------------------------------------------------------*/
#define FEATURE_FWUPDATE                               _C("FirmwareUpdate")

/* attribute Firmware Update Buffer */
#define ATT_FWUPDATE_BUFFER                            _C("Buffer")
/* VAL_FWUPDATE_BUFFER                                 %s */

/* attribute Firmware Update Size */
#define ATT_FWUPDATE_SIZE                              _C("Size")
/* VAL_FWUPDATE_SIZE                                   %d */

/* attribute Firmware Update Continue */
#define ATT_FWUPDATE_CONTINUE                          _C("Continue")
/* #define VAL_ON                                      _C("On")
   #define VAL_OFF                                     _C("Off") */

/* attribute Firmware Update Progress */
#define ATT_FWUPDATE_PROGRESS                          _C("Progress")
/* VAL_FWUPDATE_PROGRESS (0% to 100%)                  %d */

/* attribute Firmware Update Version */
#define ATT_FWUPDATE_VERSION                           _C("Version")
/* VAL_FWUPDATE_VERSION                                %s */

/* attribute Firmware Success (optional; if set FW update is completed */
#define ATT_FWUPDATE_SUCCESS                           _C("Success")
/* #define VAL_ON                                      _C("On")
   #define VAL_OFF                                     _C("Off") */

/*---------------------------------------------------------------------
      Memory Dump
----------------------------------------------------------------------*/
#define FEATURE_MEMDUMP                                _C("MemoryDump")

/* attribute Memory Dump Buffer */
#define ATT_MEMDUMP_BUFFER                             _C("Buffer")
/* VAL_MEMDUMP_BUFFER                                  %s */

/* attribute Memory Dump Size */
#define ATT_MEMDUMP_SIZE                               _C("Size")
/* VAL_MEMDUMP_SIZE                                    %d */

/* attribute Memory Dump Continue */
#define ATT_MEMDUMP_CONTINUE                           _C("Continue")
/* #define VAL_ON                                      _C("On")
   #define VAL_OFF                                     _C("Off") */

/* attribute Memory Dump Progress */
#define ATT_MEMDUMP_PROGRESS _C("Progress")
/* VAL_MEMDUMP_PROGRESS (0% to 100%)                   %d */
/* attribute Memory Dump Success (optional; set this attribute only if memory
dump is completed/aborted  */
#define ATT_MEMDUMP_SUCCESS                        _C("Success")
/* #define VAL_ON                                  _C("On")
   #define VAL_OFF                                 _C("Off") */

/*---------------------------------------------------------------------
      Diagnostic_Ex Information
---------------------------------------------------------------------*/
#define FEATURE_DIAGNOSTIC_EX                      _C("Diagnostic_Ex")

/* attribute Language */
#define ATT_DIAG_LANGUAGE_EX                       _C("Language")
/* VAL_DIAG_LANGUAGE                               %s */

/* attribute Diagnostic entry count */
#define ATT_DIAG_COUNT_EX                          _C("Count")
/* VAL_DIAG_COUNT                                  %d */

/* attribute buffer */
#define ATT_DIAG_BUFFER                            _C("Buffer")
/* VAL_DIAG_BUFFER                                 %s */

/* VAL_DIAG_BUFFER %s
   Start BP3552: 10/10/2003
   each entry in the buffer is separated by "@" sign
   the entry order is fixed as below
   0. Time
   1. Date
   2. EventShort
   3. EventLong
   4. EventID
   5. EventHex
   End BP3552 : 10/10/2003 */

#ifdef __cplusplus
#endif

#endif
```

Visual C# .NET Sample Program

Introduction to the Visual C# .NET Demo Panel Program

As described in Introducing the Sample Programs, WinAC ODK installs a Visual C# .NET implementation of a control panel that can interact with a WinAC controller. This project is located in the ...\Program Files\Siemens\WinAC\ODK\Examples\CMI\Demo_Panel_Cs directory.

You can click the DemoDlg file in the C# Solution Explorer picture to access its code listing (online help only):



Demo

Definitions.cs

```
using System;

namespace Demo
{
	public class CDemoException : ApplicationException
	{
		public CDemoException()
			: base(null, null)
		{
		}

		public CDemoException(Exception InnerException)
			: base(null, InnerException)
		{
		}
	}

	public class CDemoWarning : ApplicationException
	{
		public CDemoWarning()
			: base(null, null)
		{
		}
	}
}
```

DemoDlg.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;
using S7WCUPIntLib;
using FEATUREPROVIDERLib;

namespace Demo
{
      /// <summary>
      /// Summary description for Form1.
      /// </summary>
      public class Form1 : Form, IFeatureCallback
      {
            // all dialog controls
            private Button m_btnConnect = null;
            private Button m_btnType = null;
            private Button m_btnRUNP = null;
            private Button m_btnRUN = null;
            private Button m_btnSTOP = null;
            private Button m_btnMRES = null;
            private Label lblInstanceName = null;
            private Label lblPlcType = null;
            private Label lblSiemens = null;
            private Label lblPs = null;
            private Label lblCpu = null;
            private TextBox m_edtInstanceName = null;
            private TextBox m_edtType = null;
            private CheckBox m_chkON = null;
            private CheckBox m_chkBATF = null;
            private CheckBox m_chkINTF = null;
            private CheckBox m_chkEXTF = null;
            private CheckBox m_chkBUSF1 = null;
            private CheckBox m_chkBUSF2 = null;
            private CheckBox m_chkFRCE = null;
            private CheckBox m_chkRUN = null;
            private CheckBox m_chkSTOP = null;
            private CheckBox m_chkRUNPbtn = null;
            private CheckBox m_chkRUNbtn = null;
            private CheckBox m_chkSTOPbtn = null;

            // notification IDs
            private int m_ConnectionNotifyID = -1;
            private int m_KeyswitchNotifyID = -1;
            private int m_LedNotifyID = -1;
            private int m_PlcNotifyID = -1;
            private int m_ErrorNotifyID = -1;

            // interface pointer
            private IPLC      m_pIPlc = null;
            private IFeature m_pIFeature = null;
```

```
        // available features
        private string[] m_saFeatureNames = null;

        private const string BTN_TXT_CONNECT = "Connect";
        private const string BTN_TXT_DISCONNECT = "Disconnect";

        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;

        public Form1()
        {
            // Required for Windows Form Designer support
            InitializeComponent();

            // start an instance of the feature provider
            m_pIPlc = new PLCClass();

            // set default instance name
            m_edtInstanceName.Text = "WinLC";

            // clear plc type info
            m_edtType.Text = "";

            // set the initial connect button text
            m_btnConnect.Text = BTN_TXT_CONNECT;

            // disable all panel controls
            this.DeactivateControls();
       }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
               }
            }

            // disconnect from PLC
            this.Disconnect();

            base.Dispose( disposing );
       }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
```

```
            {
                  this.m_btnConnect = new Button();
                  this.m_btnType = new Button();
                  this.m_btnRUNP = new Button();
                  this.m_btnRUN = new Button();
                  this.m_btnSTOP = new Button();
                  this.m_btnMRES = new Button();
                  this.lblInstanceName = new Label();
                  this.lblPlcType = new Label();
                  this.lblSiemens = new Label();
                  this.lblPs = new Label();
                  this.lblCpu = new Label();
                  this.m_edtInstanceName = new TextBox();
                  this.m_edtType = new TextBox();
                  this.m_chkON = new CheckBox();
                  this.m_chkBATF = new CheckBox();
                  this.m_chkINTF = new CheckBox();
                  this.m_chkEXTF = new CheckBox();
                  this.m_chkBUSF1 = new CheckBox();
                  this.m_chkBUSF2 = new CheckBox();
                  this.m_chkFRCE = new CheckBox();
                  this.m_chkRUN = new CheckBox();
                  this.m_chkSTOP = new CheckBox();
                  this.m_chkRUNPbtn = new CheckBox();
                  this.m_chkRUNbtn = new CheckBox();
                  this.m_chkSTOPbtn = new CheckBox();
                  this.SuspendLayout();
                  //
                  // m_btnConnect
                  //
                  this.m_btnConnect.Location = new System.Drawing.Point(100,
25);
                  this.m_btnConnect.Name = "m_btnConnect";
                  this.m_btnConnect.Size = new System.Drawing.Size(70, 21);
                  this.m_btnConnect.TabIndex = 1;
                  this.m_btnConnect.Text = "Connect";
                  this.m_btnConnect.Click += new
System.EventHandler(this.m_btnConnect_Click);
                  //
                  // m_btnType
                  //
                  this.m_btnType.Location = new System.Drawing.Point(100, 67);
                  this.m_btnType.Name = "m_btnType";
                  this.m_btnType.Size = new System.Drawing.Size(70, 21);
                  this.m_btnType.TabIndex = 2;
                  this.m_btnType.Text = "Get Type";
                  this.m_btnType.Click += new
System.EventHandler(this.m_btnType_Click);
                  //
                  // m_btnRUNP
                  //
                  this.m_btnRUNP.Location = new System.Drawing.Point(95, 193);
                  this.m_btnRUNP.Name = "m_btnRUNP";
                  this.m_btnRUNP.Size = new System.Drawing.Size(48, 18);
                  this.m_btnRUNP.TabIndex = 3;
                  this.m_btnRUNP.Text = "RUN-P";
                  this.m_btnRUNP.Click += new
```

```
System.EventHandler(this.m_btnRUNP_Click);
                  //
                  // m_btnRUN
                  //
                  this.m_btnRUN.Location = new System.Drawing.Point(95, 213);
                  this.m_btnRUN.Name = "m_btnRUN";
                  this.m_btnRUN.Size = new System.Drawing.Size(48, 18);
                  this.m_btnRUN.TabIndex = 4;
                  this.m_btnRUN.Text = "RUN";
                  this.m_btnRUN.Click += new
System.EventHandler(this.m_btnRUN_Click);
                  //
                  // m_btnSTOP
                  //
                  this.m_btnSTOP.Location = new System.Drawing.Point(95, 232);
                  this.m_btnSTOP.Name = "m_btnSTOP";
                  this.m_btnSTOP.Size = new System.Drawing.Size(48, 18);
                  this.m_btnSTOP.TabIndex = 5;
                  this.m_btnSTOP.Text = "STOP";
                  this.m_btnSTOP.Click += new
System.EventHandler(this.m_btnSTOP_Click);
                  //
                  // m_btnMRES
                  //
                  this.m_btnMRES.Location = new System.Drawing.Point(95, 307);
                  this.m_btnMRES.Name = "m_btnMRES";
                  this.m_btnMRES.Size = new System.Drawing.Size(48, 18);
                  this.m_btnMRES.TabIndex = 6;
                  this.m_btnMRES.Text = "MRES";
                  this.m_btnMRES.Click += new
System.EventHandler(this.m_btnMRES_Click);
                  //
                  // lblInstanceName
                  //
                  this.lblInstanceName.Location = new System.Drawing.Point(10,
10);
                  this.lblInstanceName.Name = "lblInstanceName";
                  this.lblInstanceName.Size = new System.Drawing.Size(86, 15);
                  this.lblInstanceName.TabIndex = 0;
                  this.lblInstanceName.Text = "Instance name:";
                  //
                  // lblPlcType
                  //
                  this.lblPlcType.Location = new System.Drawing.Point(10, 51);
                  this.lblPlcType.Name = "lblPlcType";
                  this.lblPlcType.Size = new System.Drawing.Size(86, 15);
                  this.lblPlcType.TabIndex = 0;
                  this.lblPlcType.Text = "PLC type:";
                  //
                  // lblSiemens
                  //
                  this.lblSiemens.Location = new System.Drawing.Point(10, 111);
                  this.lblSiemens.Name = "lblSiemens";
                  this.lblSiemens.Size = new System.Drawing.Size(60, 15);
                  this.lblSiemens.TabIndex = 0;
                  this.lblSiemens.Text = "SIEMENS";
                  //
```

```
                        // lblPs
                        //
                        this.lblPs.Location = new System.Drawing.Point(10, 127);
                        this.lblPs.Name = "lblPs";
                        this.lblPs.Size = new System.Drawing.Size(60, 15);
                        this.lblPs.TabIndex = 0;
                        this.lblPs.Text = "PS";
                        //
                        // lblCpu
                        //
                        this.lblCpu.Location = new System.Drawing.Point(10, 187);
                        this.lblCpu.Name = "lblCpu";
                        this.lblCpu.Size = new System.Drawing.Size(60, 15);
                        this.lblCpu.TabIndex = 0;
                        this.lblCpu.Text = "CPU";
                        //
                        // m_edtInstanceName
                        //
                        this.m_edtInstanceName.Location = new
System.Drawing.Point(10, 25);
                        this.m_edtInstanceName.Name = "m_edtInstanceName";
                        this.m_edtInstanceName.Size = new System.Drawing.Size(86,
20);
                        this.m_edtInstanceName.TabIndex = 0;
                        this.m_edtInstanceName.Text = "";
                        //
                        // m_edtType
                        //
                        this.m_edtType.Enabled = false;
                        this.m_edtType.Location = new System.Drawing.Point(10, 67);
                        this.m_edtType.Name = "m_edtType";
                        this.m_edtType.Size = new System.Drawing.Size(86, 20);
                        this.m_edtType.TabIndex = 0;
                        this.m_edtType.TabStop = false;
                        this.m_edtType.Text = "";
                        //
                        // m_chkON
                        //
                        this.m_chkON.Enabled = false;
                        this.m_chkON.Location = new System.Drawing.Point(22, 145);
                        this.m_chkON.Name = "m_chkON";
                        this.m_chkON.Size = new System.Drawing.Size(60, 15);
                        this.m_chkON.TabIndex = 0;
                        this.m_chkON.TabStop = false;
                        this.m_chkON.Text = "  ON";
                        //
                        // m_chkBATF
                        //
                        this.m_chkBATF.Enabled = false;
                        this.m_chkBATF.Location = new System.Drawing.Point(22, 163);
                        this.m_chkBATF.Name = "m_chkBATF";
                        this.m_chkBATF.Size = new System.Drawing.Size(60, 15);
                        this.m_chkBATF.TabIndex = 0;
                        this.m_chkBATF.TabStop = false;
                        this.m_chkBATF.Text = "  BATF";
                        //
                        // m_chkINTF
```

```
//
this.m_chkINTF.Enabled = false;
this.m_chkINTF.Location = new System.Drawing.Point(22, 202);
this.m_chkINTF.Name = "m_chkINTF";
this.m_chkINTF.Size = new System.Drawing.Size(60, 15);
this.m_chkINTF.TabIndex = 0;
this.m_chkINTF.TabStop = false;
this.m_chkINTF.Text = "  INTF";
//
// m_chkEXTF
//
this.m_chkEXTF.Enabled = false;
this.m_chkEXTF.Location = new System.Drawing.Point(22, 220);
this.m_chkEXTF.Name = "m_chkEXTF";
this.m_chkEXTF.Size = new System.Drawing.Size(60, 15);
this.m_chkEXTF.TabIndex = 0;
this.m_chkEXTF.TabStop = false;
this.m_chkEXTF.Text = "  EXTF";
//
// m_chkBUSF1
//
this.m_chkBUSF1.Enabled = false;
this.m_chkBUSF1.Location = new System.Drawing.Point(22, 238);
this.m_chkBUSF1.Name = "m_chkBUSF1";
this.m_chkBUSF1.Size = new System.Drawing.Size(60, 15);
this.m_chkBUSF1.TabIndex = 0;
this.m_chkBUSF1.TabStop = false;
this.m_chkBUSF1.Text = "  BUSF1";
//
// m_chkBUSF2
//
this.m_chkBUSF2.Enabled = false;
this.m_chkBUSF2.Location = new System.Drawing.Point(22, 256);
this.m_chkBUSF2.Name = "m_chkBUSF2";
this.m_chkBUSF2.Size = new System.Drawing.Size(60, 15);
this.m_chkBUSF2.TabIndex = 0;
this.m_chkBUSF2.TabStop = false;
this.m_chkBUSF2.Text = "  BUSF2";
//
// m_chkFRCE
//
this.m_chkFRCE.Enabled = false;
this.m_chkFRCE.Location = new System.Drawing.Point(22, 274);
this.m_chkFRCE.Name = "m_chkFRCE";
this.m_chkFRCE.Size = new System.Drawing.Size(60, 15);
this.m_chkFRCE.TabIndex = 0;
this.m_chkFRCE.TabStop = false;
this.m_chkFRCE.Text = "  FRCE";
//
// m_chkRUN
//
this.m_chkRUN.Enabled = false;
this.m_chkRUN.Location = new System.Drawing.Point(22, 292);
this.m_chkRUN.Name = "m_chkRUN";
this.m_chkRUN.Size = new System.Drawing.Size(60, 15);
this.m_chkRUN.TabIndex = 0;
this.m_chkRUN.TabStop = false;
```

```
                  this.m_chkRUN.Text = "  RUN";
                  //
                  // m_chkSTOP
                  //
                  this.m_chkSTOP.Enabled = false;
                  this.m_chkSTOP.Location = new System.Drawing.Point(22, 310);
                  this.m_chkSTOP.Name = "m_chkSTOP";
                  this.m_chkSTOP.Size = new System.Drawing.Size(60, 15);
                  this.m_chkSTOP.TabIndex = 0;
                  this.m_chkSTOP.TabStop = false;
                  this.m_chkSTOP.Text = "  STOP";
                  //
                  // m_chkRUNPbtn
                  //
                  this.m_chkRUNPbtn.Enabled = false;
                  this.m_chkRUNPbtn.Location = new System.Drawing.Point(147,
195);
                  this.m_chkRUNPbtn.Name = "m_chkRUNPbtn";
                  this.m_chkRUNPbtn.Size = new System.Drawing.Size(20, 15);
                  this.m_chkRUNPbtn.TabIndex = 0;
                  this.m_chkRUNPbtn.TabStop = false;
                  //
                  // m_chkRUNbtn
                  //
                  this.m_chkRUNbtn.Enabled = false;
                  this.m_chkRUNbtn.Location = new System.Drawing.Point(147,
215);
                  this.m_chkRUNbtn.Name = "m_chkRUNbtn";
                  this.m_chkRUNbtn.Size = new System.Drawing.Size(20, 15);
                  this.m_chkRUNbtn.TabIndex = 0;
                  this.m_chkRUNbtn.TabStop = false;
                  //
                  // m_chkSTOPbtn
                  //
                  this.m_chkSTOPbtn.Enabled = false;
                  this.m_chkSTOPbtn.Location = new System.Drawing.Point(147,
234);
                  this.m_chkSTOPbtn.Name = "m_chkSTOPbtn";
                  this.m_chkSTOPbtn.Size = new System.Drawing.Size(20, 15);
                  this.m_chkSTOPbtn.TabIndex = 0;
                  this.m_chkSTOPbtn.TabStop = false;
                  //
                  // Form1
                  //
                  this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
                  this.ClientSize = new System.Drawing.Size(181, 351);
                  this.Controls.AddRange(new Control[]{
                  this.FormBorderStyle = FormBorderStyle.FixedDialog;
                  this.MaximizeBox = false;
                  this.MinimizeBox = false;
                  this.Name = "Form1";
                  this.Text = "Demo";
                  this.ResumeLayout(false);

           }
            #endregion
            /// <summary>
            /// The main entry point for the application.
            /// </summary>
            [STAThread]
            static void Main()
            {
                  Application.Run(new Form1());
            }

            public void OnFeatureChanged(string FeatureName, object Context,
int NotificationID, object AttributeNames, object AttributeValues)
            {
                  // init
                  string strName;
                  string strValue;
                  string[] saNames;
                  string[] saValues;

                  // get the arrays
                  saNames =  (string[])AttributeNames;
                  saValues = (string[])AttributeValues;

                  // if it's the keyswitch feature...
                  if(FeatureName.CompareTo(Feature.FEATURE_KEYSWITCH) == 0)
                  {
                        // go thru all attributes...
                        for(uint i=0; i<saNames.GetLength(0); i++)
                        {
                              // get the attribute name
                              strName = saNames[i];

                              // if it's the keyswitch attribute...
                              if(strName.CompareTo(Feature.ATT_KEYSWITCH) == 0)
                              {
                                    // get the attribute value
                                    strValue = saValues[i];

                                    // if the keyswitch is set to RUNP...
                                    if(strValue.CompareTo(Feature.VAL_KEYSWITCH
_RUNP) == 0)
                                    {
                                          m_chkRUNPbtn.Checked = true;
                                          m_chkRUNbtn.Checked = false;
                                          m_chkSTOPbtn.Checked = false;
                                    }
                                          // if the keyswitch is set to RUN...
                                    else
if(strValue.CompareTo(Feature.VAL_KEYSWITCH_RUN) == 0)
                                    {
                                          m_chkRUNPbtn.Checked = false;
                                          m_chkRUNbtn.Checked = true;
                                          m_chkSTOPbtn.Checked = false;
                                    }
                                          // if the keyswitch is set to STOP...
                                    else
if(strValue.CompareTo(Feature.VAL_KEYSWITCH_STOP) == 0)
                                    {
```

```
                                                m_chkRUNPbtn.Checked = false;
                                                m_chkRUNbtn.Checked = false;
                                                m_chkSTOPbtn.Checked = true;
                                         }
                                    }
                              }
                        }
                              // if it's the led feature...
                        else if(FeatureName.CompareTo(Feature.FEATURE_LED) == 0)
                        {
                              // go thru all attributes...
                              for(uint i=0; i<saNames.GetLength(0); i++)
                              {
                                    // get the attribute name
                                    strName = saNames[i];

                                    // if it's the power led attribute...
                                    if(strName.CompareTo(Feature.ATT_LED_POWER) == 0)
                                    {
                                          // get the attribute value
                                          strValue = saValues[i];

                                          // if the led is off...
                                          if(strValue.CompareTo(Feature.VAL_LED_OFF)
== 0)
                                                m_chkON.Checked = false;
                                                // if it is not off it is on (we
don't show blinking LEDs)
                                          else
                                                m_chkON.Checked = true;
                                    }
                                          // if it's the batf led attribute...
                                    else if(strName.CompareTo(Feature.ATT_LED_BATF)
== 0)
                                    {
                                          // get the attribute value
                                          strValue = saValues[i];

                                          // if the led is off...
                                          if(strValue.CompareTo(Feature.VAL_LED_OFF)
== 0)
                                                m_chkBATF.Checked = false;
                                                // if it is not off it is on (we
don't show blinking LEDs)
                                          else
                                                m_chkBATF.Checked = true;
                                    }
                                          // if it's the intf led attribute...
                                    else if(strName.CompareTo(Feature.ATT_LED_INTF)
== 0)
                                    {
                                          // get the attribute value
                                          strValue = saValues[i];

                                          // if the led is off...
                                          if(strValue.CompareTo(Feature.VAL_LED_OFF)
== 0)
```

```
                                        m_chkINTF.Checked = false;
                                        // if it is not off it is on (we
don't show blinking LEDs)
                                else
                                        m_chkINTF.Checked = true;
                        }
                                // if it's the extf led attribute...
                        else if(strName.CompareTo(Feature.ATT_LED_EXTF) ==
0)
                        {
                                // get the attribute value
                                strValue = saValues[i];

                                // if the led is off...
                                if(strValue.CompareTo(Feature.VAL_LED_OFF)
== 0)
                                        m_chkEXTF.Checked = false;
                                        // if it is not off it is on (we
don't show blinking LEDs)
                                else
                                        m_chkEXTF.Checked = true;
                        }
                                // if it's the busf1 led attribute...
                        else if(strName.CompareTo(Feature.ATT_LED_BUSF +
"0") == 0)
                        {
                                // get the attribute value
                                strValue = saValues[i];

                                // if the led is off...
                                if(strValue.CompareTo(Feature.VAL_LED_OFF)
== 0)
                                        m_chkBUSF1.Checked = false;
                                        // if it is not off it is on (we
don't show blinking LEDs)
                                else
                                        m_chkBUSF1.Checked = true;
                        }
                                // if it's the busf2 led attribute...
                        else if(strName.CompareTo(Feature.ATT_LED_BUSF +
"1") == 0)
                        {
                                // get the attribute value
                                strValue = saValues[i];

                                // if the led is off...
                                if(strValue.CompareTo(Feature.VAL_LED_OFF)
== 0)
                                        m_chkBUSF2.Checked = false;
                                        // if it is not off it is on (we
don't show blinking LEDs)
                                else
                                        m_chkBUSF2.Checked = true;
                        }
                                // if it's the frce led attribute...
                        else if(strName.CompareTo(Feature.ATT_LED_FORCE)
== 0)
```

```
                              {
                                    // get the attribute value
                                    strValue = saValues[i];

                                    // if the led is off...
                                    if(strValue.CompareTo(Feature.VAL_LED_OFF)
== 0)
                                          m_chkFRCE.Checked = false;
                                          // if it is not off it is on (we
don't show blinking LEDs)
                                    else
                                          m_chkFRCE.Checked = true;
                              }
                                    // if it's the run led attribute...
                              else if(strName.CompareTo(Feature.ATT_LED_RUN) ==
0)
                              {
                                    // get the attribute value
                                    strValue = saValues[i];

                                    // if the led is off...
                                    if(strValue.CompareTo(Feature.VAL_LED_OFF)
== 0)
                                          m_chkRUN.Checked = false;
                                          // if it is not off it is on (we
don't show blinking LEDs)
                                    else
                                          m_chkRUN.Checked = true;
                              }
                                    // if it's the stop led attribute...
                              else if(strName.CompareTo(Feature.ATT_LED_STOP)
== 0)
                              {
                                    // get the attribute value
                                    strValue = saValues[i];

                                    // if the led is off...
                                    if(strValue.CompareTo(Feature.VAL_LED_OFF)
== 0)
                                          m_chkSTOP.Checked = false;
                                          // if it is not off it is on (we
don't show blinking LEDs)
                                    else
                                          m_chkSTOP.Checked = true;
                              }
                        }
                  }
                        // if it's the plc feature...
                  else if(FeatureName.CompareTo(Feature.FEATURE_PLC) == 0)
                  {
                        // go thru all attributes...
                        for(uint i=0; i<saNames.GetLength(0); i++)
                        {
                              // get the attribute name
                              strName = saNames[i];

                              // if it's the plc attribute...
```

```
                    if(strName.CompareTo(Feature.ATT_PLC) == 0)
                    {
                        // get the attribute value
                        strValue = saValues[i];

                        // if the PLC is created...
                        if(strValue.CompareTo(Feature.VAL_PLC_CREAT
ED) == 0)
                            ActivateControls();
                            // if the PLC not running...
                        else
                            DeactivateControls();
                    }
                }
            }
                // if it's the error feature...
            else if(FeatureName.CompareTo(Feature.FEATURE_ERROR) == 0)
            {
                // go thru all attributes...
                for(uint i=0; i<saNames.GetLength(0); i++)
                {
                    // get the attribute name
                    strName = saNames[i];

                    // if it's the error id attribute...
                    if(strName.CompareTo(Feature.ATT_ERROR_ID) == 0)
                    {
                        // get the attribute value
                        strValue = saValues[i];

                        // if it's an error....
                        if((Feature.VAL_ERROR_ID)Convert.ToInt32(st
rValue) != Feature.VAL_ERROR_ID._PSERR_OKAY)
                        {
                            // create message
                            string strMessage;
                            strMessage = "The FeatureProvider
returned an error.";

                            // show error message
                            MessageBox.Show(strMessage, "Error");
                        }
                    }
                }
            }
        }  // end of OnFeatureChanged

        public void OnPLCDisconnect(int ErrorID)
        {
            // set the application as disconnected
            ConnectionLost();
        }

        private void Connect()
        {
            // init
            string strInstanceName;
```

```
                    object saAttributeNames = null;

                    // get the instance name
                    strInstanceName = m_edtInstanceName.Text;

                    // disable the instance name box
                    m_edtInstanceName.Enabled = false;

                    // connect to the PLC
                    this.ConnectToPlc(strInstanceName, ref m_pIFeature, ref
m_saFeatureNames, ref saAttributeNames);

                    // register all necessary features
                    this.RegisterAllFeatures();
            }

             private void Disconnect()
             {
                    // unregister all features
                    this.UnregisterAllFeatures();

                    // clear the plc proxy
                    this.ClearPlcData();

                    // deactivate all controls
                    this.DeactivateControls();

                    // clear the plc type info
                    m_edtType.Text = "";

                    // enable the instance name box
                    m_edtInstanceName.Enabled = true;
            }

             private void ConnectionLost()
             {
                    // clear the plc proxy
                    this.ClearPlcData();

                    // deactivate all controls
                    this.DeactivateControls();

                    // change the connect button
                    m_btnConnect.Text = BTN_TXT_CONNECT;

                    // disable the connect button => app needs to be restartet
                    m_btnConnect.Enabled = false;

                    // clear the plc type info
                    m_edtType.Text = "";
            }

             private void DeactivateControls()
             {
                    // uncheck all radio buttons
                    m_chkON.Checked = false;
                    m_chkBATF.Checked = false;
```

```
                  m_chkINTF.Checked = false;
                  m_chkEXTF.Checked = false;
                  m_chkBUSF1.Checked = false;
                  m_chkBUSF2.Checked = false;
                  m_chkFRCE.Checked = false;
                  m_chkRUN.Checked = false;
                  m_chkSTOP.Checked = false;
                  m_chkRUNPbtn.Checked = false;
                  m_chkRUNbtn.Checked = false;
                  m_chkSTOPbtn.Checked = false;

                  // deactivate buttons
                  m_btnRUNP.Enabled = false;
                  m_btnRUN.Enabled = false;
                  m_btnSTOP.Enabled = false;
                  m_btnMRES.Enabled = false;
                  m_btnType.Enabled = false;
            }

            private void ActivateControls()
            {
                  // activate buttons
                  m_btnRUNP.Enabled = true;
                  m_btnRUN.Enabled = true;
                  m_btnSTOP.Enabled = true;
                  m_btnMRES.Enabled = true;
                  m_btnType.Enabled = true;
            }

            private void Browse(ref string[] saConnectStrings, ref string[]
saStartInfos)
            {
                  // clear the connection string array
                  saConnectStrings = new string[0];
                  object ConnectStrings = saConnectStrings;

                  // clear the start infos array
                  saStartInfos = new string[0];
                  object StartInfos = saStartInfos;

                  // init
                  int ErrorID = 0;

                  // browse for configured and/or running PLCs
                  m_pIPlc.Browse(ref ConnectStrings, ref StartInfos, ref
ErrorID);

                  // error?
                  if(ErrorID != 0)throw new CDemoException();

                  // return values
                  saConnectStrings = (string[])ConnectStrings;
                  saStartInfos = (string[])StartInfos;
            }

            private void ConnectToPlc(string InstanceName, ref IFeature
pIFeature, ref string[] saFeatureNames, ref object saAttributeNames)
```

```
            {
                  // init
                  string strConnectionString;
                  string[] saConnectStrings = null;
                  string[] saStartInfos = null;

                  // get connection strings for all available PLCs
                  this.Browse(ref saConnectStrings, ref saStartInfos);

                  // go thru all returned connection strings and
                  // look for the requested instance name
                  for(uint i=0; i<saConnectStrings.GetLength(0); i++)
                  {
                        // get connection string
                        strConnectionString = saConnectStrings[i];

                        // extract the instance name out of the connection
string
                        int pos = strConnectionString.IndexOf("\\", 0);
                        int len = strConnectionString.IndexOf("\\", pos + 1) -
pos - 1;
                        strConnectionString = strConnectionString.Substring(pos

+ 1, len);

                        // if we have found the correct connection string...
                        if(strConnectionString.CompareTo(InstanceName) == 0)
                        {
                              // get connection string
                              strConnectionString = saConnectStrings[i];

                              // init
                              int ErrorID = 0;
                              pIFeature = null;
                              saFeatureNames = new string[0];
                              object FeatureNames = saFeatureNames;
                              saAttributeNames = 0;

                              // connect to the PLC
                              m_pIPlc.Connect(strConnectionString, ref
pIFeature, ref FeatureNames, ref saAttributeNames, ref ErrorID);

                              // error?
                              if(ErrorID != 0)throw new CDemoException();

                              // return values
                              saFeatureNames = (string[])FeatureNames;

                              // we are done
                             return;
                        }
                  }

                  // we couldn't connect
                  throw new CDemoException();
            }

            private void ClearPlcData()
            {
                  m_ConnectionNotifyID = -1;
                  m_KeyswitchNotifyID  = -1;
                  m_LedNotifyID        = -1;
                  m_PlcNotifyID        = -1;
                  m_ErrorNotifyID      = -1;

                  // release the controller management interface
                  if(m_pIFeature != null)
                  {
                        Marshal.Release(Marshal.GetIUnknownForObject(m_pIFeatur
e));

                        m_pIFeature = null;
                  }
           }

            private void RegisterAllFeatures()
            {
                  // error
                  if(m_pIFeature == null)throw new CDemoException();

                  // init
                  int ErrorID = 0;
                  string strFeatureName;

                  // register to get a notification when the connection to the
PLC is lost
                  m_pIFeature.RegisterForConnectionCheck(this, ref
m_ConnectionNotifyID, ref ErrorID);

                  // error?
                  if(ErrorID != 0)throw new CDemoException();

                  // go thru all
features.....
                  for(uint i=0; i<m_saFeatureNames.GetLength(0);
i++)
                  {

                        //
get the feature name
                        strFeatureName =
m_saFeatureNames[i];


                        // if it's one of the features we
need...
                        if(strFeatureName.CompareTo(Feature.FEATURE_KEYSWITCH)
== 0)
                        {

                              //
...register it
                              this.RegisterFeature(strFeatureName, ref
m_KeyswitchNotifyID);
                       }
```

```
                        else if(strFeatureName.CompareTo(Feature.FEATURE_LED)
== 0)
                        {

                              //
...register it
                              this.RegisterFeature(strFeatureName, ref
m_LedNotifyID);
                       }

                        else if(strFeatureName.CompareTo(Feature.FEATURE_PLC)
== 0)
                        {

                              //
...register it
                              this.RegisterFeature(strFeatureName, ref
m_PlcNotifyID);
                       }

                        else if(strFeatureName.CompareTo(Feature.FEATURE_ERROR)
== 0)
                        {

                              //
...register it
                              this.RegisterFeature(strFeatureName, ref
m_ErrorNotifyID);
                       }

                  }
           }

            private void UnregisterAllFeatures()
            {
                  // error
                  if(m_pIFeature == null)return;

                  // let's unregister the features => the feature provider
stopps sending notifications
                  if(m_KeyswitchNotifyID != -1)
                        this.UnregisterFeature(Feature.FEATURE_KEYSWITCH,
m_KeyswitchNotifyID);
                  if(m_LedNotifyID != -1)
                        this.UnregisterFeature(Feature.FEATURE_LED,
m_LedNotifyID);
                  if(m_PlcNotifyID != -1)
                        this.UnregisterFeature(Feature.FEATURE_PLC,
m_PlcNotifyID);
                  if(m_ErrorNotifyID != -1)
                        this.UnregisterFeature(Feature.FEATURE_ERROR,
m_ErrorNotifyID);

                  // unregister for connection check
                  if(m_ConnectionNotifyID != -1)
                  {
                        int ErrorID = 0;
                        m_pIFeature.UnregisterForConnectionCheck(m_ConnectionNo
tifyID, ref ErrorID);
                  }
           }

            private void RegisterFeature(string FeatureName, ref int
NotificationID)
            {
                  // error
                  if(m_pIFeature == null)throw new CDemoException();

                  // init
                  int ErrorID = 0;
                  object context;

                  // create a context variable (what ever...)
                  context = "DemoApp";

                  // register the feature to get change notifications
                  m_pIFeature.RegisterFeatureForChange(this, FeatureName,
context, ref NotificationID, ref ErrorID);

                  // error?
                  if(ErrorID != 0)throw new CDemoException();
           }

            private void UnregisterFeature(string FeatureName, int
NotificationID)
            {
                  // error
                  if(m_pIFeature == null)throw new CDemoException();

                  // init
                  int ErrorID = 0;

                  // unregister the feature to cancel notifications
                  m_pIFeature.UnregisterFeatureForChange(NotificationID, ref
ErrorID);

                  // error?
                  if(ErrorID != 0)throw new CDemoException();
           }

            private void SetKeyswitch(string Value)
            {
                  // error
                  if(m_pIFeature == null)throw new CDemoException();

                  // init
                  int ErrorID = 0;
                  string[] saAttributeNames;
                  string[] saAttributeValues;

                  // create arrays
                  saAttributeNames = new string[1];
                  saAttributeValues = new string[1];
                  // set values
                  saAttributeNames[0] = Feature.ATT_KEYSWITCH;
                  saAttributeValues[0] = Value;

                  // set the keyswitch
                  m_pIFeature.SetFeature(Feature.FEATURE_KEYSWITCH,
saAttributeNames, saAttributeValues, ref ErrorID);

                  // error?
                  if(ErrorID != 0)throw new CDemoException();
           }

            private void GetCpuType(ref string Value)
            {
                  // error
                  if(m_pIFeature == null)throw new CDemoException();

                  // init
                  Value = "";
                  int ErrorID = 0;
                  string[] saAttributeNames = new string[0];
                  string[] saAttributeValues = new string[0];
                  object AttributeNames = saAttributeNames;
                  object AttributeValues = saAttributeValues;

                  // get the personality feature attribute values
                  m_pIFeature.GetFeature(Feature.FEATURE_PERSONALITY, ref
AttributeNames, ref AttributeValues, ref ErrorID);

                  // error?
                  if(ErrorID != 0)throw new CDemoException();

                  // get values
                  saAttributeNames = (string[])AttributeNames;
                  saAttributeValues = (string[])AttributeValues;

                  // loop thru all attributes
                  for(uint i=0; i<saAttributeNames.GetLength(0); i++)
                  {
                        // if it's the plc type attribute
                        if(saAttributeNames[i].CompareTo(Feature.ATT_PERSONALIT
Y_TYPE) == 0)
                        {
                              // get type
                              Value = saAttributeValues[i];
                             return;
                       }
                  }
                  // the plc type attribute was not provided
                  throw new CDemoWarning();
           }

            private void m_btnConnect_Click(object sender, System.EventArgs e)
            {
                  // lock the button
                  m_btnConnect.Enabled = false;
                  // init
                  string strBtnText;

                  try
                  {
                        // get the button text
                        strBtnText = m_btnConnect.Text;

                        // if we want to connect...
                        if(strBtnText.CompareTo(BTN_TXT_CONNECT) == 0)
                        {
                              try
                              {
                                    // connect to the PLC
                                    this.Connect();

                                    // change the connect button
                                    m_btnConnect.Text = BTN_TXT_DISCONNECT;
                              }
                              catch(Exception)
                              {
                                    // disconnect the PLC and clear all
settings
                                    this.Disconnect();

                                    // show error message box
                                    MessageBox.Show("Connecting to the PLC
failed.", "Error");
                              }
                       }
                        // if we want to disconnect...
                        else if(strBtnText.CompareTo(BTN_TXT_DISCONNECT) == 0)
                        {
                              // disconnect the PLC
                              this.Disconnect();

                              // change the connect button
                              m_btnConnect.Text = BTN_TXT_CONNECT;
                       }
                  }
                  catch(Exception)
                  {
                  }

                  // unlock the button
                  m_btnConnect.Enabled = true;
           }

            private void m_btnType_Click(object sender, System.EventArgs e)
            {
                  // init
                  string strPlcType = "";

                  try
                  {
                        // get the plc type info
                        this.GetCpuType(ref strPlcType);
```

```
        }
        catch(Exception exc)
        {
              // if plc type is not provided....
              if(exc.GetType() == typeof(CDemoWarning))
                    strPlcType = "unknown";
              // if it's an error....
              else
                    strPlcType = "error";
        }

        // display the plc type
        m_edtType.Text = strPlcType;
 }

 private void m_btnRUNP_Click(object sender, System.EventArgs e)
 {
        try
        {
              // send a request to change the keyswitch to RUN-P
              this.SetKeyswitch(Feature.VAL_KEYSWITCH_RUNP);
        }
        catch(Exception)
        {
        }
 }

 private void m_btnRUN_Click(object sender, System.EventArgs e)
 {
        try
        {
              // send a request to change the keyswitch to RUN
              this.SetKeyswitch(Feature.VAL_KEYSWITCH_RUN);
        }
        catch(Exception)
        {
        }
 }

 private void m_btnSTOP_Click(object sender, System.EventArgs e)
 {
        try
        {
              // send a request to change the keyswitch to STOP
              this.SetKeyswitch(Feature.VAL_KEYSWITCH_STOP);
        }
        catch(Exception)
        {
        }
 }

 private void m_btnMRES_Click(object sender, System.EventArgs e)
 {
        try
        {
              // create message
              string strMessage = "The module will be reset
```

```
(clear/reset).  All user data will be deleted and all existing connections to
the module will be disconnected.  Do you really want to reset the module?";

                        // double-check with the user
                        if(MessageBox.Show(null, strMessage, "Demo",
MessageBoxButtons.YesNo, MessageBoxIcon.Warning,
MessageBoxDefaultButton.Button1) == DialogResult.Yes)
                        {
                              // send a request for MRES
                              this.SetKeyswitch(Feature.VAL_KEYSWITCH_MRES);
                       }
                  }
                  catch(Exception)
                  {
                  }
           }
      }
}
```

Feature.cs

```
using System;

namespace Demo
{
      //Here are all constants defined which we need for our work
      //with the feature provider (the list is not complete).

      struct Feature
      {
            //----------------------------------------------------------------
            //  PLC
            //----------------------------------------------------------------
            public const string FEATURE_PLC = "PLC";

            public const string ATT_PLC = "Value";
            public const string VAL_PLC_CREATED = "Created";
            public const string VAL_PLC_DESTROYED = "Destroyed";

            //----------------------------------------------------------------
            //  LED
            //----------------------------------------------------------------
            public const string FEATURE_LED = "LED";

            //attributes
            public const string ATT_LED_POWER = "Power";
            public const string ATT_LED_BATF = "BatteryFault";
            public const string ATT_LED_INTF = "InternalFault";
            public const string ATT_LED_EXTF = "ExternalFault";
            public const string ATT_LED_BUSFAULTCOUNT = "BusFaultCount";
            public const string ATT_LED_BUSF = "BusFault_";
            public const string ATT_LED_FORCE = "Force";
            public const string ATT_LED_RUN = "Run";
            public const string ATT_LED_STOP = "Stop";

            //values
            public const string VAL_LED_ON = "ON";
            public const string VAL_LED_OFF = "OFF";
            public const string VAL_LED_BLINKING2HZ = "Blinking2HZ";
            public const string VAL_LED_BLINKING05HZ = "Blinking05HZ";

            //----------------------------------------------------------------
            //  KeySwitch
            //----------------------------------------------------------------
            public const string FEATURE_KEYSWITCH = "KeySwitch";

            //attribute KeySwitch
            public const string ATT_KEYSWITCH = "Value";
            public const string VAL_KEYSWITCH_MRES = "MRES";
            public const string VAL_KEYSWITCH_STOP = "STOP";
            public const string VAL_KEYSWITCH_RUN = "RUN";
            public const string VAL_KEYSWITCH_RUNP = "RUNP";

            //attribute force coldstart
            public const string ATT_KEYSWITCH_FORCECOLDSTART =
"ForceColdstart";
```

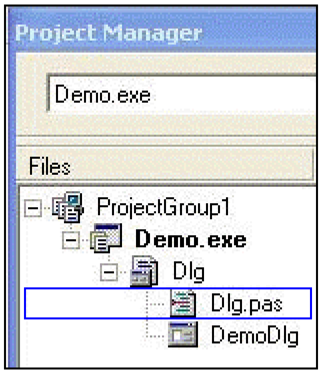
```
            //public const string VAL_ON = "On"
            public const string VAL_OFF = "Off";

            //----------------------------------------------------------------
            //  Personality
            //----------------------------------------------------------------
            public const string FEATURE_PERSONALITY = "Personality";

            //attributes
            public const string ATT_PERSONALITY_NAME = "Name";
            public const string ATT_PERSONALITY_TYPE = "Type";
            public const string ATT_PERSONALITY_PRODUCTCODE = "ProductCode";
            public const string ATT_PERSONALITY_SW_RELEASE = "SWRelease";
            public const string ATT_PERSONALITY_FW_RELEASE = "FWRelease";
            public const string ATT_PERSONALITY_HW_RELEASE = "HWRelease";
            public const string ATT_PERSONALITY_SLOT_NUMBER = "Slot";
            public const string ATT_PERSONALITY_RACK_NUMBER = "Rack";
            public const string ATT_PERSONALITY_OWNER_INFO = "Owner";
            public const string ATT_PERSONALITY_COMPANY_INFO = "Company";

            //----------------------------------------------------------------
            //  Error
            //----------------------------------------------------------------
            public const string FEATURE_ERROR = "Error";

            // attribute error identification
            public const string ATT_ERROR_ID = "ID";

            // valid error identifications
            public enum PSERR
            {
                  PSERR_OKAY = 0,
                  PSERR_NO_MEMORY,
                  PSERR_ARCHIVE_NOT_VALID_IN_RUN,
                  PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU,
                  PSERR_RESTORE_CANNOT_LINKIN_BLOCK,
                  PSERR_RESTORE_NOT_VALID_IN_RUN,
                  PSERR_RESTORE_FILE_INVALID,
                  PSERR_INIT_EDBSERVER,
                  PSERR_INIT_PDH,
                  PSERR_KEYSWITCH_NOT_ALLOWED_IN_MCF_OP,
                  PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU_PASSWORD_PROTECTED
            }
      }
}
```

Borland Delphi 7 Sample Program

Introduction to the Borland Delphi 7 Demo Panel Program

As described in Introducing the Sample Programs, WinAC ODK installs a Borland Delphi 7 implementation of a control panel that can interact with a WinAC controller. This project is located in the ...\Program Files\Siemens\WinAC\ODK\Examples\CMI\Demo_Panel_Delphi directory.

You can click the Dlg.pas file in the Delphi 7 Project Manager picture to access its code listing (online help only):



Demo

Dlg.pas

```
unit Dlg;

interface

uses
   Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
   ComObj, S7UP_Interfaces, S7UP_FeatureProvider, S7UP_Feature, StdCtrls;

type
  EDemoException = class(EComponentError);

type
  EDemoWarning = class(EComponentError);

type
  TDemoDlg = class(TForm, IFeatureCallback)
    lblInstanceName: TLabel;
    edtInstanceName: TEdit;
    lblType: TLabel;
    edtType: TEdit;
    lblSiemens: TLabel;
    lblPs: TLabel;
    chkON: TCheckBox;
    chkBATF: TCheckBox;
    lblCpu: TLabel;
    chkINTF: TCheckBox;
    chkEXTF: TCheckBox;
    chkBUSF1: TCheckBox;
    chkBUSF2: TCheckBox;
    chkFRCE: TCheckBox;
    chkRUN: TCheckBox;
    chkSTOP: TCheckBox;
    btnConnect: TButton;
    btnType: TButton;
    btnRUNP: TButton;
    btnRUN: TButton;
    btnSTOP: TButton;
    btnMRES: TButton;
    chkRUNPbtn: TCheckBox;
    chkRUNbtn: TCheckBox;
    chkSTOPbtn: TCheckBox;
    procedure FormDestroy(Sender: TObject);
    procedure btnConnectClick(Sender: TObject);
    procedure btnTypeClick(Sender: TObject);
    procedure btnRUNPClick(Sender: TObject);
    procedure btnRUNClick(Sender: TObject);
    procedure btnSTOPClick(Sender: TObject);
    procedure btnMRESClick(Sender: TObject);
  private
    procedure ActivateControls;
    procedure DeactivateControls;
    procedure Connect;
    procedure Disconnect;
    procedure ConnectionLost;
    procedure Browse(var ConnectStrings: OleVariant; var StartInfos:
OleVariant);
    procedure ConnectToPlc(var InstanceName: String; var pIFeature: IFeature;
                           var FeatureNames: OleVariant; var AttributeNames:
OleVariant);
    procedure ClearPlcData;
    procedure RegisterAllFeatures;
    procedure UnregisterAllFeatures;
    procedure RegisterFeature(var FeatureName: WideString; var pNotificationID:
LongInt);
    procedure UnregisterFeature(var FeatureName: WideString; var
NotificationID: LongInt);
    procedure SetKeyswitch(var Value: WideString);
    procedure GetCpuType(var pValue: WideString);
  protected
    procedure Loaded; override;
  public
    destructor Destroy; override;
    procedure OnFeatureChanged(const FeatureName: WideString; Context:
OleVariant;
                               NotificationID: Integer; AttributeNames:
OleVariant;
                               AttributeValues: OleVariant); safecall;
    procedure OnPLCDisconnect(ErrorID: Integer); safecall;
  end;

const
  BTN_TXT_CONNECT: String = 'Connect';
  BTN_TXT_DISCONNECT: String = 'Disconnect';

var
  DemoDlg: TDemoDlg;
  pIPLC: IPLC;
  pIFeature: IFeature;
  FeatureNames: OleVariant;
  ConnectionNotifyID: LongInt;
  KeyswitchNotifyID: LongInt;
  LedNotifyID: LongInt;
  PlcNotifyID: LongInt;
  ErrorNotifyID: LongInt;

implementation

uses Variants, StrUtils;

destructor TDemoDlg.Destroy;
begin
  inherited Destroy;
end;

procedure TDemoDlg.Loaded;
begin
  // start COM
  CoInitializeEx(nil, 0);

  // create an instance of the feature provider
  pIPLC := CoPLC.Create;

  // initialize
  Self.ClearPlcData;

  // set default instance name
  edtInstanceName.Text := 'WinLC';

  // clear plc type info
  edtType.Text := '';

  // set the initial connect button text
  btnConnect.Caption := BTN_TXT_CONNECT;

  // disable all panel controls
  Self.DeactivateControls;
end;

procedure TDemoDlg.FormDestroy(Sender: TObject);
begin
  // disconnect
  Self.Disconnect;
end;

procedure TDemoDlg.OnFeatureChanged(const FeatureName: WideString; Context:
OleVariant;
                               NotificationID: Integer; AttributeNames:
OleVariant;
                               AttributeValues: OleVariant);
var
  i: Integer;
  strName: WideString;
  strValue: WideString;
begin
  // if it's the keyswitch feature...
  if WideCompareStr(FeatureName, FEATURE_KEYSWITCH) = 0 then
  begin
    // go thru all attributes...
    for i := VarArrayLowBound(AttributeNames, 1) to
VarArrayHighBound(AttributeNames, 1) do
    begin
      // get the attribute name
      strName := AttributeNames[i];

      // if it's the keyswitch attribute...
      if WideCompareStr(strName, ATT_KEYSWITCH) = 0 then
                  begin
        // get the attribute value
        strValue := AttributeValues[i];

        // if the keyswitch is set to RUNP...
        if WideCompareStr(strValue, VAL_KEYSWITCH_RUNP) = 0 then
        begin
          chkRUNPbtn.Checked := true;
          chkRUNbtn.Checked := false;
          chkSTOPbtn.Checked := false;
        end
        // if the keyswitch is set to RUN...
        else if WideCompareStr(strValue, VAL_KEYSWITCH_RUN) = 0 then
        begin
          chkRUNPbtn.Checked := false;
          chkRUNbtn.Checked := true;
          chkSTOPbtn.Checked := false;
        end
        // if the keyswitch is set to STOP...
        else if WideCompareStr(strValue, VAL_KEYSWITCH_STOP) = 0 then
        begin
          chkRUNPbtn.Checked := false;
          chkRUNbtn.Checked := false;
          chkSTOPbtn.Checked := true;
        end
      end;
    end;
  end
  // if it's the led feature...
  else if WideCompareStr(FeatureName, FEATURE_LED) = 0 then
  begin
    // go thru all attributes...
    for i := VarArrayLowBound(AttributeNames, 1) to
VarArrayHighBound(AttributeNames, 1) do
    begin
      // get the attribute name
      strName := AttributeNames[i];

      // if it's the power led attribute...
      if WideCompareStr(strName, ATT_LED_POWER) = 0 then
      begin
        // get the attribute value
        strValue := AttributeValues[i];

        // if the led is off...
        if WideCompareStr(strValue, VAL_LED_OFF) = 0 then
          chkON.Checked := false
        // if it is not off it is on (we don't show blinking LEDs)
        else
          chkON.Checked := true;
      end
      // if it's the batf led attribute...
      else if WideCompareStr(strName, ATT_LED_BATF) = 0 then
      begin
        // get the attribute value
        strValue := AttributeValues[i];

        // if the led is off...
        if WideCompareStr(strValue, VAL_LED_OFF) = 0 then
          chkBATF.Checked := false
        // if it is not off it is on (we don't show blinking LEDs)
        else
          chkBATF.Checked := true;
      end
      // if it's the intf led attribute...
      else if WideCompareStr(strName, ATT_LED_INTF) = 0 then
      begin
        // get the attribute value
        strValue := AttributeValues[i];

        // if the led is off...
        if WideCompareStr(strValue, VAL_LED_OFF) = 0 then
          chkINTF.Checked := false
        // if it is not off it is on (we don't show blinking LEDs)
        else
          chkINTF.Checked := true;
      end
      // if it's the extf led attribute...
      else if WideCompareStr(strName, ATT_LED_EXTF) = 0 then
      begin
        // get the attribute value
        strValue := AttributeValues[i];

        // if the led is off...
        if WideCompareStr(strValue, VAL_LED_OFF) = 0 then
          chkEXTF.Checked := false
        // if it is not off it is on (we don't show blinking LEDs)
        else
          chkEXTF.Checked := true;
      end
      // if it's the busf1 led attribute...
      else if WideCompareStr(strName, ATT_LED_BUSF + '0') = 0 then
      begin
        // get the attribute value
        strValue := AttributeValues[i];

        // if the led is off...
        if WideCompareStr(strValue, VAL_LED_OFF) = 0 then
          chkBUSF1.Checked := false
        // if it is not off it is on (we don't show blinking LEDs)
        else
          chkBUSF1.Checked := true;
      end
      // if it's the busf2 led attribute...
      else if WideCompareStr(strName, ATT_LED_BUSF + '1') = 0 then
      begin
        // get the attribute value
        strValue := AttributeValues[i];

        // if the led is off...
        if WideCompareStr(strValue, VAL_LED_OFF) = 0 then
          chkBUSF2.Checked := false
        // if it is not off it is on (we don't show blinking LEDs)
        else
          chkBUSF2.Checked := true;
      end
      // if it's the frce led attribute...
      else if WideCompareStr(strName, ATT_LED_FORCE) = 0 then
      begin
        // get the attribute value
        strValue := AttributeValues[i];

        // if the led is off...
        if WideCompareStr(strValue, VAL_LED_OFF) = 0 then
          chkFRCE.Checked := false
```

```
          // if it is not off it is on (we don't show blinking LEDs)
          else
            chkFRCE.Checked := true;
        end
        // if it's the run led attribute...
        else if WideCompareStr(strName, ATT_LED_RUN) = 0 then
        begin
          // get the attribute value
          strValue := AttributeValues[i];

          // if the led is off...
          if WideCompareStr(strValue, VAL_LED_OFF) = 0 then
            chkRUN.Checked := false
          // if it is not off it is on (we don't show blinking LEDs)
          else
            chkRUN.Checked := true;
        end
        // if it's the stop led attribute...
        else if WideCompareStr(strName, ATT_LED_STOP) = 0 then
        begin
          // get the attribute value
          strValue := AttributeValues[i];

          // if the led is off...
          if WideCompareStr(strValue, VAL_LED_OFF) = 0 then
            chkSTOP.Checked := false
          // if it is not off it is on (we don't show blinking LEDs)
          else
            chkSTOP.Checked := true;
        end;
      end;
    end
    // if it's the plc feature...
    else if WideCompareStr(FeatureName, FEATURE_PLC) = 0 then
    begin
      // go thru all attributes...
      for i := VarArrayLowBound(AttributeNames, 1) to
VarArrayHighBound(AttributeNames, 1) do
      begin
        // get the attribute name
        strName := AttributeNames[i];

        // if it's the plc attribute...
        if WideCompareStr(strName, ATT_PLC) = 0 then
        begin
          // get the attribute value
          strValue := AttributeValues[i];

          // if the PLC is created...
          if WideCompareStr(strValue, VAL_PLC_CREATED) = 0 then
            ActivateControls()
          // if the PLC not running...
          else
            DeactivateControls();
        end;
      end;
    end
```

```
  // if it's the error feature...
  else if WideCompareStr(FeatureName, FEATURE_ERROR) = 0 then
  begin
    // go thru all attributes...
    for i := VarArrayLowBound(AttributeNames, 1) to
VarArrayHighBound(AttributeNames, 1) do
    begin
      // get the attribute name
      strName := AttributeNames[i];

      // if it's the error id attribute...
      if WideCompareStr(strName, ATT_ERROR_ID) = 0 then
      begin
        // get the attribute value
        strValue := AttributeValues[i];

        // if it's an error....
        if StrToInt(strValue) <> _PSERR_OKAY then
          // show error message
          MessageBox(Self.Handle, 'The FeatureProvider returned an error.',
'Error', MB_OK);
      end;
    end;
  end;
end;

procedure TDemoDlg.OnPLCDisconnect(ErrorID: Integer);
begin
  // set the application as disconnected
  Self.ConnectionLost;
end;

procedure TDemoDlg.ActivateControls;
begin
  // activate buttons
  btnRUNP.Enabled := true;
  btnRUN.Enabled := true;
  btnSTOP.Enabled := true;
  btnMRES.Enabled := true;
  btnType.Enabled := true;
end;

procedure TDemoDlg.DeactivateControls;
begin
  // uncheck all radio buttons
  chkON.Checked := false;
  chkBATF.Checked := false;
  chkINTF.Checked := false;
  chkEXTF.Checked := false;
  chkBUSF1.Checked := false;
  chkBUSF2.Checked := false;
  chkFRCE.Checked := false;
  chkRUN.Checked := false;
  chkSTOP.Checked := false;
  chkRUNPbtn.Checked := false;
  chkRUNbtn.Checked := false;
  chkSTOPbtn.Checked := false;
  // deactivate buttons
  btnRUNP.Enabled := false;
  btnRUN.Enabled := false;
  btnSTOP.Enabled := false;
  btnMRES.Enabled := false;
  btnType.Enabled := false;
end;

procedure TDemoDlg.Connect;
var
  strInstanceName: String;
  AttributeNames: OleVariant;
begin
  // get the instance name
  strInstanceName := edtInstanceName.Text;

  // disable the instance name box
  edtInstanceName.Enabled := false;

  // connect to the PLC
  Self.ConnectToPlc(strInstanceName, pIFeature, FeatureNames, AttributeNames);

  // register all necessary features
  Self.RegisterAllFeatures;
end;

procedure TDemoDlg.Disconnect;
begin
  // unregister all features
  Self.UnregisterAllFeatures;

  // clear the plc proxy
  Self.ClearPlcData;

  // deactivate all controls
  Self.DeactivateControls;

  // clear the plc type info
  edtType.Text := '';

  // enable the instance name box
  edtInstanceName.Enabled := true;
end;

procedure TDemoDlg.ConnectionLost;
begin
  // clear the plc proxy
  Self.ClearPlcData;

  // deactivate all controls
  Self.DeactivateControls;

  // change the connect button
  btnConnect.Caption := BTN_TXT_CONNECT;

  // disable the connect button => app needs to be restarted
  btnConnect.Enabled := false;

  // clear the plc type info
  edtType.Text := '';
end;

procedure TDemoDlg.Browse(var ConnectStrings: OleVariant; var StartInfos:
OleVariant);
var
  ErrorID: LongInt;
begin
  // clear the arrays
  VarClear(ConnectStrings);
  VarClear(StartInfos);

  // browse for configured and/or running PLCs
  pIPlc.Browse(ConnectStrings, StartInfos, ErrorID);

  // error?
  if(ErrorID <> 0)then raise EDemoException.Create('');
end;

procedure TDemoDlg.ConnectToPlc(var InstanceName: String; var pIFeature:
IFeature);
                           var FeatureNames: OleVariant; var AttributeNames:
OleVariant);
var
  ConnectStrings: OleVariant;
  StartInfos: OleVariant;
  ConnectString: String;
  i: Integer;
  pos: Integer;
  len: Integer;
  pErrorID: LongInt;
begin
  // get connection strings for all available PLCs
  Self.Browse(ConnectStrings, StartInfos);

  // go thru all returned connection strings and
  // look for the requested instance name
  for i := VarArrayLowBound(ConnectStrings, 1) to
VarArrayHighBound(ConnectStrings, 1) do
      begin
    // get connection string
    ConnectString := ConnectStrings[i];

    // extract the instance name out of the connection string
    pos := AnsiPos('\', ConnectString);
    ConnectString := AnsiRightStr(ConnectString, Length(ConnectString) - pos);
    len := AnsiPos('\', ConnectString) - 1;
    ConnectString := AnsiMidStr(ConnectStrings[i], pos + 1, len);

    // if we have found the correct connection string...
    if ConnectString = InstanceName then
    begin
      // get connection string
      ConnectString := ConnectStrings[i];
      // init
      pIFeature := nil;
      VarClear(FeatureNames);
      VarClear(AttributeNames);

      // connect to the PLC
      pIPlc.Connect(ConnectString, pIFeature, FeatureNames, AttributeNames,
pErrorID);

      // error?
      if pErrorID <> 0 then raise EDemoException.Create('');

      // we are done
      Exit;
    end;
  end;

  // we couldn't connect
  raise EDemoException.Create('');
end;

procedure TDemoDlg.ClearPlcData;
begin
  // reset
  ConnectionNotifyID := -1;
  KeyswitchNotifyID  := -1;
  LedNotifyID        := -1;
  PlcNotifyID        := -1;
  ErrorNotifyID      := -1;

  // release the controller management interface
  if pIFeature <> nil then
      begin
        Finalize(pIFeature);
      end;
end;

procedure TDemoDlg.RegisterAllFeatures;
var
  pErrorID: LongInt;
  strFeatureName: WideString;
  i: Integer;
begin
  // error
  if pIFeature = nil then raise EDemoException.Create('');;

  // register to get a notification when the connection to the PLC is lost
  pIFeature.RegisterForConnectionCheck(Self, ConnectionNotifyID, pErrorID);

  // error?
  if pErrorID <> 0 then raise EDemoException.Create('');

  // go thru all features.....
  for i := VarArrayLowBound(FeatureNames, 1) to
           VarArrayHighBound(FeatureNames, 1) do
      begin
    // get the feature name
    strFeatureName := FeatureNames[i];

    // if it's one of the features we need...
    if WideSameText(strFeatureName, FEATURE_KEYSWITCH) then

      // ...register it
      Self.RegisterFeature(strFeatureName, KeyswitchNotifyID)

    else if WideSameText(strFeatureName, FEATURE_LED) then

      // ...register it
      Self.RegisterFeature(strFeatureName, LedNotifyID)

    else if WideSameText(strFeatureName, FEATURE_PLC) then

      // ...register it
      Self.RegisterFeature(strFeatureName, PlcNotifyID)

    else if WideSameText(strFeatureName, FEATURE_ERROR) then

      // ...register it
      Self.RegisterFeature(strFeatureName, ErrorNotifyID)
  end;
end;

procedure TDemoDlg.UnregisterAllFeatures;
var
  pErrorID: LongInt;
begin
  // error
  if pIFeature = nil then Exit;

  // let's unregister the features => the feature provider stops sending
notifications
  if KeyswitchNotifyID <> -1 then
    Self.UnregisterFeature(FEATURE_KEYSWITCH, KeyswitchNotifyID);
  if LedNotifyID <> -1 then
    Self.UnregisterFeature(FEATURE_LED, LedNotifyID);
  if PlcNotifyID <> -1 then
    Self.UnregisterFeature(FEATURE_PLC, PlcNotifyID);
  if ErrorNotifyID <> -1 then
    Self.UnregisterFeature(FEATURE_ERROR, ErrorNotifyID);

  // unregister for connection check
  if ConnectionNotifyID <> -1 then
    pIFeature.UnregisterForConnectionCheck(ConnectionNotifyID, pErrorID);
end;

procedure TDemoDlg.RegisterFeature(var FeatureName: WideString; var
pNotificationID: LongInt);
var
  pErrorID: LongInt;
  context: OleVariant;
begin
  // error
  if pIFeature = nil then raise EDemoException.Create('');
  // create a context variable (what ever...)
  context := 'DemoApp';

  // register the feature to get change notifications
  pIFeature.RegisterFeatureForChange(Self, FeatureName, context,
pNotificationID, pErrorID);

  // error?
  if pErrorID <> 0 then raise EDemoException.Create('');
end;

procedure TDemoDlg.UnregisterFeature(var FeatureName: WideString; var
NotificationID: LongInt);
var
  pErrorID: LongInt;
begin
  // error
  if pIFeature = nil then raise EDemoException.Create('');

  // unregister the feature to cancel notifications
      pIFeature.UnregisterFeatureForChange(NotificationID, pErrorID);

  // error?
  if pErrorID <> 0 then raise EDemoException.Create('');
end;

procedure TDemoDlg.SetKeyswitch(var Value: WideString);
var
  pErrorID: LongInt;
  AttributeNames: OleVariant;
  AttributeValues: OleVariant;
begin
  // error
      if pIFeature = nil then raise EDemoException.Create('');

  // create arrays
  AttributeValues := VarArrayCreate([0,0], varOleStr);
  AttributeNames := VarArrayCreate([0,0], varOleStr);

  // set values
  AttributeNames[0] := ATT_KEYSWITCH;
  AttributeValues[0] := Value;

  // set the keyswitch
      pIFeature.SetFeature(FEATURE_KEYSWITCH, AttributeNames, AttributeValues,
pErrorID);

  // error?
      if pErrorID <> 0 then raise EDemoException.Create('');
end;

procedure TDemoDlg.GetCpuType(var pValue: WideString);
var
  pErrorID: LongInt;
  AttributeNames: OleVariant;
  AttributeValues: OleVariant;
  i: Integer;
begin
  // error
  if pIFeature = nil then raise EDemoException.Create('');

  // init
  pValue := '';

  // get the personality feature attribute values
  pIFeature.GetFeature(FEATURE_PERSONALITY, AttributeNames, AttributeValues,
pErrorID);

  // error?
  if pErrorID <> 0 then raise EDemoException.Create('');

  // loop thru all attributes
  for i := VarArrayLowBound(AttributeNames, 1) to
VarArrayHighBound(AttributeNames, 1) do
  begin
    // if it's the plc type attribute
    if WideCompareStr(AttributeNames[i], ATT_PERSONALITY_TYPE) = 0 then
    begin
      // get type
      pValue := AttributeValues[i];
      Exit;
    end;
  end;
  // the plc type attribute was not provided
  raise EDemoWarning.Create('');
end;

{$R *.DFM}

procedure TDemoDlg.btnConnectClick(Sender: TObject);
var
  strBtnText: String;
begin
  // lock the button
  btnConnect.Enabled := false;

  try
    // get the button text
    strBtnText := btnConnect.Caption;

    // if we want to connect...
    if strBtnText = BTN_TXT_CONNECT then
    begin
      try
        // connect to the PLC
        Self.Connect;

        // change the connect button
        btnConnect.Caption := BTN_TXT_DISCONNECT;
      except
        // disconnect the PLC and clear all settings
        Self.Disconnect;
        // show error message box
        MessageBox(Self.Handle, 'Connecting to the PLC failed.', 'Error', MB_OK
+ MB_ICONWARNING);
      end;
    end
    // if we want to disconnect...
    else if strBtnText = BTN_TXT_DISCONNECT then
    begin
      // disconnect the PLC
      Self.Disconnect;

      // change the connect button
      btnConnect.Caption := BTN_TXT_CONNECT;
    end
  except

  end;

  // unlock the button
  btnConnect.Enabled := true;
end;

procedure TDemoDlg.btnTypeClick(Sender: TObject);
var
  strPlcType: WideString;
begin
  try
    // get the plc type info
    Self.GetCpuType(strPlcType);
  except
    on Ex: EDemoWarning do      // if plc type is not provided....
      strPlcType := 'unknown';
    else                        // if it's an error....
      strPlcType := 'error';
  end;

                  // display the plc type
                  edtType.Text := strPlcType;
end;

procedure TDemoDlg.btnRUNPClick(Sender: TObject);
begin
  try
    // send a request to change the keyswitch to RUN-P
    Self.SetKeyswitch(VAL_KEYSWITCH_RUNP);
  except
      end;
end;

procedure TDemoDlg.btnRUNClick(Sender: TObject);
begin
  try
    // send a request to change the keyswitch to RUN
    Self.SetKeyswitch(VAL_KEYSWITCH_RUN);
  except
      end;
end;
```

```
procedure TDemoDlg.btnSTOPClick(Sender: TObject);
begin
  try
    // send a request to change the keyswitch to STOP
    Self.SetKeyswitch(VAL_KEYSWITCH_STOP);
  except
      end;
end;

procedure TDemoDlg.btnMRESClick(Sender: TObject);
var
 strMessage: WideString;
begin
  try
    // create message
    strMessage := 'The module will be reset (clear/reset). All user data will
be deleted and all existing connections to the module will be disconnected. Do
you really want to reset the module?';

    // double-check with the user
    if MessageDlg(strMessage, mtWarning, [mbYes, mbNo], 0) = mrYes then
    begin
      // send a request to change the keyswitch to MRES
      Self.SetKeyswitch(VAL_KEYSWITCH_MRES);
    end;
  except
      end;
end;

end.
```

Support

S7UP_Feature.pas

```
unit S7UP_Feature;

interface

const
//------------------------------------------------------------------
//   LED
//------------------------------------------------------------------
FEATURE_LED                       : WideString = 'LED';

// attribute power LED
ATT_LED_POWER                     : WideString = 'Power';

// attribute battery fault LED
ATT_LED_BATF                      : WideString = 'BatteryFault';

// attribute internal fault LED
ATT_LED_INTF                      : WideString = 'InternalFault';

// attribute external fault LED
ATT_LED_EXTF                      : WideString = 'ExternalFault';

// attribute bus fault LED
ATT_LED_BUSFAULTCOUNT             : WideString = 'BusFaultCount';
// VAL_LED_BUSFAULTCOUNT          %d
ATT_LED_BUSF                      : WideString = 'BusFault_';

// attribute force LED
ATT_LED_FORCE                     : WideString = 'Force';

// attribute run LED
ATT_LED_RUN                       : WideString = 'Run';

// attribute stop LED
ATT_LED_STOP                      : WideString = 'Stop';

// value LED
VAL_LED_ON                        : WideString = 'ON';
VAL_LED_OFF                       : WideString = 'OFF';
VAL_LED_BLINKING2HZ               : WideString = 'Blinking2HZ';
VAL_LED_BLINKING05HZ              : WideString = 'Blinking05HZ';

//------------------------------------------------------------------
//    KeySwitch
//------------------------------------------------------------------

FEATURE_KEYSWITCH                  : WideString = 'KeySwitch';

// attribute KeySwitch
ATT_KEYSWITCH                      : WideString = 'Value';
VAL_KEYSWITCH_MRES                 : WideString = 'MRES';
VAL_KEYSWITCH_STOP                 : WideString = 'STOP';
```

```
VAL_KEYSWITCH_RUN                   : WideString = 'RUN';
VAL_KEYSWITCH_RUNP                  : WideString = 'RUNP';

// attribute force coldstart
ATT_KEYSWITCH_FORCECOLDSTART        : WideString = 'ForceColdstart';
// VAL_ON                           : WideString = 'On';
VAL_OFF                             : WideString = 'Off';

//-------------------------------------------------------------------
//    PLCInstance
//-------------------------------------------------------------------
FEATURE_PLCINSTANCE                 : WideString = 'PLCInstance';

// attribute PLCINSTANCE
ATT_PLCINSTANCE                     : WideString = 'Value';
VAL_PLCINSTANCE_CREATE              : WideString = 'Create';
VAL_PLCINSTANCE_SHUTDOWN            : WideString = 'Shutdown';

//-------------------------------------------------------------------
//    PLCPower
//-------------------------------------------------------------------
FEATURE_PLCPOWER                    : WideString = 'PLCPower';

// attribute PLCPower state
ATT_PLCPOWER                        : WideString = 'Value';
VAL_ON                              : WideString = 'On';
// VAL_OFF                          : WideString = 'Off';

//-------------------------------------------------------------------
//    PLC
//-------------------------------------------------------------------
FEATURE_PLC                         : WideString = 'PLC';

// attribute PLC
ATT_PLC                             : WideString = 'Value';
VAL_PLC_CREATED                     : WideString = 'Created';
VAL_PLC_SHUTDOWNED                  : WideString = 'Shutdowned';
VAL_PLC_NOTAVAILABLE                : WideString = 'NotAvailable';

//-------------------------------------------------------------------
//    StartAtBoot
//-------------------------------------------------------------------
FEATURE_STARTATBOOT                 : WideString = 'StartAtBoot';

// attribute Start at boot
ATT_STARTATBOOT                     : WideString = 'Value';
// VAL_ON                           : WideString = 'On';
// VAL_OFF                          : WideString = 'Off';

//-------------------------------------------------------------------
//    AutoStart
//-------------------------------------------------------------------
FEATURE_AUTOSTART                    : WideString = 'AutoStart';

// attribute Autostart
ATT_AUTOSTART                        : WideString = 'Value';
// VAL_ON                            : WideString = 'On';
// VAL_OFF                            : WideString = 'Off';

//------------------------------------------------------------------
//    AutoLoad
//------------------------------------------------------------------
FEATURE_AUTOLOAD                       : WideString = 'AutoLoad';

// attribute Autoload enabled
ATT_AUTOLOAD                           : WideString = 'Value';
// VAL_ON                              : WideString = 'On';
// VAL_OFF                             : WideString = 'Off';

// attribute Keyswitch state after Autoload
ATT_AUTOLOAD_KEYSWITCH                 : WideString = 'KeySwitch';
VAL_AUTOLOAD_KS_STOP                   : WideString = 'STOP';
VAL_AUTOLOAD_KS_RUN                    : WideString = 'RUN';
VAL_AUTOLOAD_KS_RUNP                   : WideString = 'RUNP';

// attribute Buffer for Autoload
ATT_AUTOLOAD_BUFFER                    : WideString = 'Buffer';
// VAL_AUTOLOAD_BUFFER                 %s

// attribute Buffersize for Autoload
ATT_AUTOLOAD_BUFFERSIZE                 : WideString = 'BufferSize';
// VAL_AUTOLOAD_BUFFERSIZE              %d

// attribute target filename for Autoload
ATT_AUTOLOAD_TARGETFILE                 : WideString = 'TargetFile';
// VAL_AUTOLOAD_TARGETFILE              %s

//------------------------------------------------------------------
//    Security
//------------------------------------------------------------------
FEATURE_SECURITY                        : WideString = 'Security';

// attribute actual Password
ATT_SECURITY_ACTPASSWORD                : WideString = 'Password';
// VAL_SECURITY_ACTPASSWORD             %s scrambled

// attribute new Password
ATT_SECURITY_NEWPASSWORD                : WideString = 'NewPassword';
// VAL_SECURITY_NEWPASSWORD             %s scrambled

// attribute Password check
ATT_SECURITY_PASSWORDCHECK              : WideString = 'Check';
VAL_SECURITY_PASSWORDCHECK_PASS         : WideString = 'Passed';
VAL_SECURITY_PASSWORDCHECK_FAILED       : WideString = 'Failed';

// attribute security level
ATT_SECURITY_LEVEL                      : WideString = 'Level';
VAL_SECURITY_LEVELPASSWORD              : WideString = 'Password';
VAL_SECURITY_LEVELPASSWORDDISABLED      : WideString = 'PasswordDisabled';
VAL_SECURITY_LEVELCONFIRMATION          : WideString = 'Confirmation';
VAL_SECURITY_LEVELNONE                  : WideString = 'None';

// attribute Password Prompt Interval hours
ATT_SECURITY_INTERVALHOURS              : WideString = 'hInterval';
```

```
// VAL_SECURITY_INTERVALHOURS                  %d

// attribute Password Prompt Interval minutes
ATT_SECURITY_INTERVALMINUTES          : WideString = 'mInterval';
// VAL_SECURITY_INTERVALMINUTES                  %d

//------------------------------------------------------------------
//    FMR
//------------------------------------------------------------------
FEATURE_FMR                           : WideString = 'FMR';

//------------------------------------------------------------------
//    MemoryCard file
//------------------------------------------------------------------
FEATURE_MCF                           : WideString = 'MemoryCardFile';

// attribute MemoryCard file Buffer
ATT_MCF_BUFFER                        : WideString = 'Buffer';
// VAL_MCF_BUFFER                     %s

// attribute MemoryCard file Size
ATT_MCF_SIZE                          : WideString = 'Size';
// VAL_MCF_SIZE                       %d

//------------------------------------------------------------------
//    Priority
//------------------------------------------------------------------
FEATURE_PRIORITY                      : WideString = 'Priority';

// attribute Priority value
ATT_PRIORITY                          : WideString = 'Value';
// VAL_PRIORITY                       %d

ATT_PRIORITY_LOWERLIMIT               : WideString = 'LowerLimit';
// VAL_PRIORITY_LOWERLIMIT            %d

ATT_PRIORITY_UPPERLIMIT               : WideString = 'UpperLimit';
// VAL_PRIORITY_UPPERLIMIT            %d

ATT_NORMAL_PRIORITY                   : WideString = 'Normal';
// VAL_NORMAL_PRIORITY                %d

ATT_CRITICAL_PRIORITY                 : WideString = 'Critical';
// VAL_CRITICAL_PRIORITY              %d

//------------------------------------------------------------------
//    Minimum Sleep Time
//------------------------------------------------------------------
FEATURE_MINSLEEPTIME                  : WideString = 'MinSleepTime';

// attribute Minimum SleepTime value
ATT_MINSLEEPTIME                      : WideString = 'Value';
// VAL_MINSLEEPTIME                   %d

ATT_MINSLEEPTIME_LOWERLIMIT           : WideString = 'LowerLimit';
// VAL_MINSLEEPTIME_LOWERLIMIT        %d
```

```
ATT_MINSLEEPTIME_UPPERLIMIT             : WideString = 'UpperLimit';
// VAL_MINSLEEPTIME_UPPERLIMIT          %d

//-------------------------------------------------------------------
//     Minimum Cycle Time
//-------------------------------------------------------------------
FEATURE_MINCYCLETIME                    : WideString = 'MinCycleTime';

// attribute Minimum CycleTime value
ATT_MINCYCLETIME                        : WideString = 'Value';
// VAL_MINCYCLETIME                     %d

ATT_MINCYCLETIME_LOWERLIMIT             : WideString = 'LowerLimit';
// VAL_MINSLEEPTIME_LOWERLIMIT          %d

ATT_MINCYCLETIME_UPPERLIMIT             : WideString = 'UpperLimit';
// VAL_MINCYCLETIME_UPPERLIMIT          %d

//-------------------------------------------------------------------
//     CPU Usage
//-------------------------------------------------------------------
FEATURE_CPUUSAGE                        : WideString = 'Usage';

// attribute Usage count
ATT_PCUSAGE                             : WideString = 'PC';
// VAL_PCUSAGE                          %d [0..100%]

// attribute PLC Usage
ATT_PLCUSAGE                            : WideString = 'PLC';
// VAL_PLCUSAGE                         %d [0..100%]

// attribute Usage count
ATT_CPUUSAGE_COUNT                      : WideString = 'CPUCount';
// VAL_CPUUSAGE_COUNT                   %d

// attribute CPU Usage values
ATT_CPUUSAGE                            : WideString = 'CPU_';
// VAL_CPUUSAGE                         %d [0..100%]

//-------------------------------------------------------------------
//     Timing
//-------------------------------------------------------------------
FEATURE_TIMING                           : WideString = 'Timing';

// attribute Timing Upper Limit
ATT_TIMING_UPPERLIMIT                    : WideString = 'UpperLimit';
// VAL_TIMING_UPPERLIMIT                 %d

// attribute CycleTime count
ATT_TIMING_CYCLETIMECOUNT                : WideString = 'CycleTimeCount';
// VAL_TIMING_CYCLETIMECOUNT             %d

// attribute CycleTime buffer
ATT_TIMING_CYCLETIMEBUFFER               : WideString = 'CycleTimeBuffer';
// VAL_TIMING_CYCLETIMEBUFFER            %d %d...

// attribute CycleTime minimum
```

```
ATT_TIMING_CYCLETIMEMIN                      : WideString = 'CycleTimeMin';
// VAL_TIMING_CYCLETIMEMIN                   %d

// attribute CycleTime maximum
ATT_TIMING_CYCLETIMEMAX                      : WideString = 'CycleTimeMax';
// VAL_TIMING_CYCLETIMEMAX                   %d

// attribute CycleTime average
ATT_TIMING_CYCLETIMEAVE                      : WideString = 'CycleTimeAverage';
// VAL_TIMING_CYCLETIMEAVE                   %d

// attribute CycleTime last
ATT_TIMING_CYCLETIMELAST                     : WideString = 'CycleTimeLast';
// VAL_TIMING_CYCLETIMELAST                  %d

// attribute ExecTime minimum
ATT_TIMING_EXECTIMEMIN                       : WideString = 'ExecTimeMin';
// VAL_TIMING_EXECTIMEMIN                    %d

// attribute ExecTime maximum
ATT_TIMING_EXECTIMEMAX                       : WideString = 'ExecTimeMax';
// VAL_TIMING_EXECTIMEMAX                    %d

// attribute ExecTime average
ATT_TIMING_EXECTIMEAVE                       : WideString = 'ExecTimeAverage';
// VAL_TIMING_EXECTIMEAVE                    %d

// attribute ExecTime last
ATT_TIMING_EXECTIMELAST                       : WideString = 'ExecTimeLast';
// VAL_TIMING_EXECTIMELAST                    %d

// attribute SleepIntervalCounter
ATT_TIMING_SLEEPINTERVALCOUNTER               : WideString = 'SleepIntervalCounter';
// VAL_TIMING_SLEEPINTERVALCOUNTER            %d

// attribute Clear
ATT_TIMING_CLEAR                              : WideString = 'Clear';
// VAL_TIMING_CLEAR                           empty

//-------------------------------------------------------------------
//    OBExecution
//-------------------------------------------------------------------
FEATURE_OBEXEC                                : WideString = 'OBExecution';

// attribute WakeInterval
ATT_OBEXEC_WAKEINTERVAL                       : WideString = 'WakeInterval';
// VAL_OBEXEC_WAKEINTERVAL                    %d

// attribute SleepInterval
ATT_OBEXEC_SLEEPINTERVAL                      : WideString = 'SleepInterval';
// VAL_OBEXEC_SLEEPINTERVAL                   %d

// attribute default WakeInterval
ATT_OBEXEC_DEFAULTWAKEINTERVAL                : WideString = 'DefaultWakeInterval';
// VAL_OBEXEC_DEFAULTWAKEINTERVAL             %d

// attribute default SleepInterval
```

```
ATT_OBEXEC_DEFAULTSLEEPINTERVAL         : WideString = 'DefaultSleepInterval';
// VAL_OBEXEC_DEFAULTSLEEPINTERVAL      %d

// attribute UpperLimit maximum execution load
ATT_OBEXEC_UPPERLIMIT                   : WideString = 'UpperLimit';
// VAL_OBEXEC_UPPERLIMIT                %d [0..100%]

// attribute Lowerimit maximum execution load
ATT_OBEXEC_LOWERLIMIT                   : WideString = 'LowerLimit';
// VAL_OBEXEC_LOWERLIMIT                %d [0..100%]

//-------------------------------------------------------------------
//    Diagnostic Language
//-------------------------------------------------------------------
FEATURE_DIAGLANGUAGE                    : WideString = 'DiagnosticLanguage';

// attribute Language count
ATT_DIAGLANGUAGE_COUNT                  : WideString = 'Count';
// VAL_DIAGLANGUAGE_COUNT               %d

// attribute Language
ATT_DIAGLANGUAGE                        : WideString = 'Language_';
VAL_LANGUAGE_GERMAN                     : WideString = 'GERMAN';
VAL_LANGUAGE_ENGLISH                    : WideString = 'ENGLISH';
VAL_LANGUAGE_FRENCH                     : WideString = 'FRENCH';
VAL_LANGUAGE_ITALIAN                    : WideString = 'ITALIAN';
VAL_LANGUAGE_SPANISH                    : WideString = 'SPANISH';
VAL_LANGUAGE_JAPANESE                   : WideString = 'JAPANESE';
VAL_LANGUAGE_CHINESE                    : WideString = 'CHINESE';

//-------------------------------------------------------------------
//    Diagnostic Information
//-------------------------------------------------------------------
FEATURE_DIAGNOSTIC                      : WideString = 'Diagnostic';

// attribute Language
ATT_DIAG_LANGUAGE                       : WideString = 'Language';
// VAL_DIAG_LANGUAGE                    %s

// attribute Diagnostic entry count
ATT_DIAG_COUNT                           : WideString = 'Count';
// VAL_DIAG_COUNT                        %d

// attribute Time
ATT_DIAG_TIME                            : WideString = 'Time_';
// VAL_DIAG_TIME                         %s

// attribute Date
ATT_DIAG_DATE                            : WideString = 'Date_';
// VAL_DIAG_DATE                         %s

// attribute Event short text
ATT_DIAG_EVENTSHORT                      : WideString = 'EventShort_';
// VAL_DIAG_EVENTSHORT                   %s

// attribute Event long text
ATT_DIAG_EVENTLONG                       : WideString = 'EventLong_';
```

```
// VAL_DIAG_EVENTLONG                      %s

// attribute Event ID
ATT_DIAG_EVENTID                           : WideString = 'EventID_';
// VAL_DIAG_EVENTID                        %s

// attribute Event hex text
ATT_DIAG_EVENTHEX                          : WideString = 'EventHex_';
// VAL_DIAG_EVENTHEX                       %s

//-------------------------------------------------------------------
//    Personality
//-------------------------------------------------------------------
FEATURE_PERSONALITY                         : WideString = 'Personality';

// attribute instance name
ATT_PERSONALITY_NAME                        : WideString = 'Name';
// VAL_PERSONALITY_NAME                     %s

// attribute CPU type
ATT_PERSONALITY_TYPE                        : WideString = 'Type';
// ATT_PERSONALITY_TYPE                     %s

// attribute product code type
ATT_PERSONALITY_PRODUCTCODE                 : WideString = 'ProductCode';
// ATT_PERSONALITY_PRODUCTCODE              %s

// attribute SW release
ATT_PERSONALITY_SW_RELEASE                  : WideString = 'SWRelease';
// ATT_PERSONALITY_SW_RELEASE               %s

// attribute FW release
ATT_PERSONALITY_FW_RELEASE                  : WideString = 'FWRelease';
// ATT_PERSONALITY_FW_RELEASE               %s

// attribute HW release
ATT_PERSONALITY_HW_RELEASE                  : WideString = 'HWRelease';
// ATT_PERSONALITY_HW_RELEASE               %s

// attribute SLOT Number
ATT_PERSONALITY_SLOT_NUMBER                 : WideString = 'Slot';
// ATT_PERSONALITY_SLOT_NUMBER              %s

// attribute RACK Number
ATT_PERSONALITY_RACK_NUMBER                 : WideString = 'Rack';
// ATT_PERSONALITY_RACK_NUMBER              %s

// attribute OWNER Info
ATT_PERSONALITY_OWNER_INFO                  : WideString = 'Owner';
// ATT_PERSONALITY_OWNER_INFO               %s

// attribute RACK Number
ATT_PERSONALITY_COMPANY_INFO                : WideString = 'Company';
// ATT_PERSONALITY_COMPANY_INFO             %s

//-------------------------------------------------------------------
//    Error
```

```
//-----------------------------------------------------------------
FEATURE_ERROR                           : WideString = 'Error';

// attribute error identification
ATT_ERROR_ID                            : WideString = 'ID';
// VAL_ERROR_ID                         %d

// valid error identifications
_PSERR_OKAY                                   : Integer = 0;
_PSERR_NO_MEMORY                              : Integer = 1;
_PSERR_ARCHIVE_NOT_VALID_IN_RUN               : Integer = 2;
_PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU      : Integer = 3;
_PSERR_RESTORE_CANNOT_LINKIN_BLOCK            : Integer = 4;
_PSERR_RESTORE_NOT_VALID_IN_RUN               : Integer = 5;
_PSERR_RESTORE_FILE_INVALID                   : Integer = 6;
_PSERR_INIT_EDBSERVER                         : Integer = 7;
_PSERR_INIT_PDH                               : Integer = 8;
_PSERR_KEYSWITCH_NOT_ALLOWED_IN_MCF_OP        : Integer = 9;
_PSERR_ARCHIVE_CANNOT_GET_BLOCK_FROM_CPU_PASSWORD_PROTECTED: Integer = 10;

//-----------------------------------------------------------------
//    HelpInfo
//-----------------------------------------------------------------
FEATURE_CONTROLLER_HELP                 : WideString = 'ControllerHelp';

// attribute host
ATT_HELP_HOST                           : WideString = 'Host';
// VAL_HELP_HOST                        %s

// attribute helpsystem
ATT_HELP_SYSTEM                         : WideString = 'HelpSystem';
VAL_HELP_SYSTEM_WIN                     : WideString = 'WinHelp';
VAL_HELP_SYSTEM_WEB                     : WideString = 'WebHelp';
VAL_HELP_SYSTEM_HTML                    : WideString = 'HTMLHelp';

// attribute document help directory; includes drive letter
ATT_HELP_DIR                            : WideString = 'HelpDir';
// VAL_HELP_DIR                         %s

// attribute Language count
ATT_HELPLANGUAGE_COUNT                  : WideString = 'Count';
// VAL_HELPLANGUAGE_COUNT               %d

// attribute Language
ATT_HELPLANGUAGE                        : WideString = 'Language_';
// VAL_LANGUAGE_GERMAN                  : WideString = 'GERMAN';
// VAL_LANGUAGE_ENGLISH                 : WideString = 'ENGLISH';
// VAL_LANGUAGE_FRENCH                  : WideString = 'FRENCH';
// VAL_LANGUAGE_ITALIAN                 : WideString = 'ITALIAN';
// VAL_LANGUAGE_SPANISH                 : WideString = 'SPANISH';
// VAL_LANGUAGE_JAPANESE                : WideString = 'JAPANESE';
// VAL_LANGUAGE_CHINESE                 : WideString = 'CHINESE';


//-----------------------------------------------------------------
//    CPU Language
//-----------------------------------------------------------------
```

```
FEATURE_CPULANGUAGE                        : WideString = 'CPULanguage';

// attribute Language
ATT_CPULANGUAGE_CURRENT                    : WideString = 'CurrentLanguage';
// VAL_LANGUAGE_GERMAN                     : WideString = 'GERMAN';
// VAL_LANGUAGE_ENGLISH                    : WideString = 'ENGLISH';
// VAL_LANGUAGE_FRENCH                     : WideString = 'FRENCH';
// VAL_LANGUAGE_ITALIAN                    : WideString = 'ITALIAN';
// VAL_LANGUAGE_SPANISH                    : WideString = 'SPANISH';
// VAL_LANGUAGE_JAPANESE                   : WideString = 'JAPANESE';
// VAL_LANGUAGE_CHINESE                    : WideString = 'CHINESE';


// attribute Language count
ATT_CPULANGUAGE_COUNT                     : WideString = 'Count';
// VAL_CPULANGUAGE_COUNT                  %d

// attribute Language
ATT_CPULANGUAGE                           : WideString = 'Language_';
// VAL_LANGUAGE_GERMAN                    : WideString = 'GERMAN';
// VAL_LANGUAGE_ENGLISH                   : WideString = 'ENGLISH';
// VAL_LANGUAGE_FRENCH                    : WideString = 'FRENCH';
// VAL_LANGUAGE_ITALIAN                   : WideString = 'ITALIAN';
// VAL_LANGUAGE_SPANISH                   : WideString = 'SPANISH';
// VAL_LANGUAGE_JAPANESE                  : WideString = 'JAPANESE';
// VAL_LANGUAGE_CHINESE                   : WideString = 'CHINESE';


//--------------------------------------------------------------------
//    PC/PG Interface
//--------------------------------------------------------------------
FEATURE_PCPGINTERFACE                     : WideString = 'PC_PG_Interface';

implementation

end.
```

S7UP_FeatureProvider.pas

```
unit S7UP_FeatureProvider;

{$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
interface

uses Windows, ActiveX, Classes, Graphics, OleServer, OleCtrls, StdVCL,
     S7UP_Interfaces;

const
  FEATUREPROVIDERLibMajorVersion = 1;
  FEATUREPROVIDERLibMinorVersion = 0;

  LIBID_FEATUREPROVIDERLib: TGUID = '{D9268C30-ABEF-4E01-AE65-5CC47520657F}';

  CLASS_PLC: TGUID = '{56FF7A88-C074-48A5-B89F-A9EF8B3B46E2}';
  CLASS_PLCConnection: TGUID = '{DBA38236-46D9-45F2-83C2-10D73F555CE9}';

type
  __MIDL___MIDL_itf_FeatureProvider_0000_0001 = TOleEnum;

const
  EFP_BASE = $00000000;
  EFP_SUCCESS = $00000000;
  EFP_INTERNAL_ERROR = $00000001;
  EFP_PARAM_INVALID = $00000002;
  EFP_NO_MEMORY = $00000003;
  EFP_FEATURE_INVALID = $00000004;
  EFP_NOTIFICATION_ID_INVALID = $00000005;
  EFP_ALREADY_REGISTERED = $00000006;
  EFP_CONNECTION_FAILURE = $00000007;
  EFP_PARSER_ERROR = $00000008;
  EFP_BROWSE_FAILED = $00000009;
  EFP_CONNECT_FAILED = $0000000A;
  EFP_DISCONNECT_FAILED = $0000000B;
  EFP_REGISTER_FEATURE_FAILED = $0000000C;
  EFP_UNREGISTER_FEATURE_FAILED = $0000000D;
  EFP_GET_FEATURE_FAILED = $0000000E;
  EFP_SET_FEATURE_FAILED = $0000000F;
  EFP_GET_CHANGED_FEATURE_FAILED = $00000010;
  EFP_CREATE_OBJECT_FAILED = $00000011;
  EFP_MULTICAST_FAILED = $00000011;

type
  PLC = IPLC;
  PLCConnection = IFeature;

  EFP_ERROR = __MIDL___MIDL_itf_FeatureProvider_0000_0001;

  CoPLC = class
    class function Create: IPLC;
    class function CreateRemote(const MachineName: string): IPLC;
  end;

// *********************************************************************//
// Server Object: TPLC
// *********************************************************************//
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
  TPLCProperties= class;
{$ENDIF}
  TPLC = class(TOleServer)
  private
    FIntf:        IPLC;
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
    FProps:       TPLCProperties;
    function      GetServerProperties: TPLCProperties;
{$ENDIF}
    function      GetDefaultInterface: IPLC;
  protected
    procedure InitServerData; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor  Destroy; override;
    procedure Connect; override;
    procedure ConnectTo(svrIntf: IPLC);
    procedure Disconnect; override;
    procedure Browse(var pConnectStrings: OleVariant; var pStartInfos:
OleVariant;
                     var pErrorID: Integer);
    procedure Connect1(const ConnectString: WideString; var pIFeature:
IFeature;
                       var pFeatureNames: OleVariant; var pAttributeNamesArray:
OleVariant;
                       var pErrorID: Integer);
    property  DefaultInterface: IPLC read GetDefaultInterface;
  published
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
    property Server: TPLCProperties read GetServerProperties;
{$ENDIF}
  end;

{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
// *********************************************************************//
// Server Object: TPLC
// *********************************************************************//
 TPLCProperties = class(TPersistent)
  private
    FServer:    TPLC;
    function    GetDefaultInterface: IPLC;
    constructor Create(AServer: TPLC);
  protected
  public
    property DefaultInterface: IPLC read GetDefaultInterface;
  published
  end;
{$ENDIF}

// *********************************************************************//
// Server Object: TPLCConnection
// *********************************************************************//
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
  TPLCConnectionProperties= class;
{$ENDIF}
```

```
  TPLCConnection = class(TOleServer)
  private
    FIntf:        IFeature;
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
    FProps:       TPLCConnectionProperties;
    function      GetServerProperties: TPLCConnectionProperties;
{$ENDIF}
    function      GetDefaultInterface: IFeature;
  protected
    procedure InitServerData; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor  Destroy; override;
    procedure Connect; override;
    procedure ConnectTo(svrIntf: IFeature);
    procedure Disconnect; override;
    procedure GetFeature(const FeatureName: WideString; var pAttributeNames:
OleVariant;
                         var pAttributeValues: OleVariant; var pErrorID:
Integer);
    procedure SetFeature(const FeatureName: WideString; AttributeNames:
OleVariant;
                         AttributeValues: OleVariant; var pErrorID: Integer);
    procedure RegisterFeatureForChange(const pCallback: IFeatureCallback;
                                       const FeatureName: WideString; Context:
OleVariant;
                                       var pNotificationID: Integer; var
pErrorID: Integer);
    procedure UnregisterFeatureForChange(NotificationID: Integer; var pErrorID:
Integer);
    procedure RegisterForConnectionCheck(const pCallback: IFeatureCallback;
                                         var pNotificationID: Integer; var
pErrorID: Integer);
    procedure UnregisterForConnectionCheck(NotificationID: Integer; var
pErrorID: Integer);
    property  DefaultInterface: IFeature read GetDefaultInterface;
  published
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
    property Server: TPLCConnectionProperties read GetServerProperties;
{$ENDIF}
  end;

{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
// *********************************************************************//
// Server Object: TPLCConnection
// *********************************************************************//
 TPLCConnectionProperties = class(TPersistent)
  private
    FServer:    TPLCConnection;
    function    GetDefaultInterface: IFeature;
    constructor Create(AServer: TPLCConnection);
  protected
  public
    property DefaultInterface: IFeature read GetDefaultInterface;
  published
  end;
{$ENDIF}
```

```
procedure Register;

implementation

uses ComObj;

class function CoPLC.Create: IPLC;
begin
  Result := CreateComObject(CLASS_PLC) as IPLC;
end;

class function CoPLC.CreateRemote(const MachineName: string): IPLC;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_PLC) as IPLC;
end;

procedure TPLC.InitServerData;
const
  CServerData: TServerData = (
    ClassID:   '{56FF7A88-C074-48A5-B89F-A9EF8B3B46E2}';
    IntfIID:   '{2F8564E3-CEC5-4747-A109-F52743F80292}';
    EventIID:  '';
    LicenseKey: nil;
    Version: 500);
begin
  ServerData := @CServerData;
end;

procedure TPLC.Connect;
var
  punk: IUnknown;
begin
  if FIntf = nil then
  begin
    punk := GetServer;
    Fintf:= punk as IPLC;
  end;
end;

procedure TPLC.ConnectTo(svrIntf: IPLC);
begin
  Disconnect;
  FIntf := svrIntf;
end;

procedure TPLC.DisConnect;
begin
  if Fintf <> nil then
  begin
    FIntf := nil;
  end;
end;

function TPLC.GetDefaultInterface: IPLC;
begin
  if FIntf = nil then
```

```
    Connect;
  Assert(FIntf <> nil, 'DefaultInterface is NULL. Component is not connected to
Server. You must call ''Connect'' or ''ConnectTo'' before this operation');
  Result := FIntf;
end;

constructor TPLC.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
  FProps := TPLCProperties.Create(Self);
{$ENDIF}
end;

destructor TPLC.Destroy;
begin
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
  FProps.Free;
{$ENDIF}
  inherited Destroy;
end;

{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
function TPLC.GetServerProperties: TPLCProperties;
begin
  Result := FProps;
end;
{$ENDIF}

procedure TPLC.Browse(var pConnectStrings: OleVariant; var pStartInfos:
OleVariant;
                      var pErrorID: Integer);
begin
  DefaultInterface.Browse(pConnectStrings, pStartInfos, pErrorID);
end;

procedure TPLC.Connect1(const ConnectString: WideString; var pIFeature:
IFeature;
                        var pFeatureNames: OleVariant; var
pAttributeNamesArray: OleVariant;
                        var pErrorID: Integer);
begin
  DefaultInterface.Connect(ConnectString, pIFeature, pFeatureNames,
pAttributeNamesArray, pErrorID);
end;

{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
constructor TPLCProperties.Create(AServer: TPLC);
begin
  inherited Create;
  FServer := AServer;
end;

function TPLCProperties.GetDefaultInterface: IPLC;
begin
  Result := FServer.DefaultInterface;
end;
```

```
{$ENDIF}

procedure TPLCConnection.InitServerData;
const
  CServerData: TServerData = (
    ClassID:   '{DBA38236-46D9-45F2-83C2-10D73F555CE9}';
    IntfIID:   '{4E00B20B-F5CA-4BEE-A599-3DE2FBED32B4}';
    EventIID:  '';
    LicenseKey: nil;
    Version: 500);
begin
  ServerData := @CServerData;
end;

procedure TPLCConnection.Connect;
var
  punk: IUnknown;
begin
  if FIntf = nil then
  begin
    punk := GetServer;
    Fintf:= punk as IFeature;
  end;
end;

procedure TPLCConnection.ConnectTo(svrIntf: IFeature);
begin
  Disconnect;
  FIntf := svrIntf;
end;

procedure TPLCConnection.DisConnect;
begin
  if Fintf <> nil then
  begin
    FIntf := nil;
  end;
end;

function TPLCConnection.GetDefaultInterface: IFeature;
begin
  if FIntf = nil then
    Connect;
  Assert(FIntf <> nil, 'DefaultInterface is NULL. Component is not connected to
Server. You must call ''Connect'' or ''ConnectTo'' before this operation');
  Result := FIntf;
end;

constructor TPLCConnection.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
  FProps := TPLCConnectionProperties.Create(Self);
{$ENDIF}
end;
```

```
destructor TPLCConnection.Destroy;
begin
{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
  FProps.Free;
{$ENDIF}
  inherited Destroy;
end;

{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
function TPLCConnection.GetServerProperties: TPLCConnectionProperties;
begin
  Result := FProps;
end;
{$ENDIF}

procedure TPLCConnection.GetFeature(const FeatureName: WideString; var
pAttributeNames: OleVariant;
                                    var pAttributeValues: OleVariant; var
pErrorID: Integer);
begin
  DefaultInterface.GetFeature(FeatureName, pAttributeNames, pAttributeValues,
pErrorID);
end;

procedure TPLCConnection.SetFeature(const FeatureName: WideString;
AttributeNames: OleVariant;
                                    AttributeValues: OleVariant; var pErrorID:
Integer);
begin
  DefaultInterface.SetFeature(FeatureName, AttributeNames, AttributeValues,
pErrorID);
end;

procedure TPLCConnection.RegisterFeatureForChange(const pCallback:
IFeatureCallback;
                                                  const FeatureName:
WideString;
                                                  Context: OleVariant;
                                                  var pNotificationID: Integer;
                                                  var pErrorID: Integer);
begin
  DefaultInterface.RegisterFeatureForChange(pCallback, FeatureName, Context,
pNotificationID,
                                            pErrorID);
end;

procedure TPLCConnection.UnregisterFeatureForChange(NotificationID: Integer;
var pErrorID: Integer);
begin
  DefaultInterface.UnregisterFeatureForChange(NotificationID, pErrorID);
end;

procedure TPLCConnection.RegisterForConnectionCheck(const pCallback:
IFeatureCallback;
                                                    var pNotificationID:
Integer;
                                                    var pErrorID: Integer);
```

```
begin
  DefaultInterface.RegisterForConnectionCheck(pCallback, pNotificationID, 
pErrorID);
end;

procedure TPLCConnection.UnregisterForConnectionCheck(NotificationID: Integer; 
var pErrorID: Integer);
begin
  DefaultInterface.UnregisterForConnectionCheck(NotificationID, pErrorID);
end;

{$IFDEF LIVE_SERVER_AT_DESIGN_TIME}
constructor TPLCConnectionProperties.Create(AServer: TPLCConnection);
begin
  inherited Create;
  FServer := AServer;
end;

function TPLCConnectionProperties.GetDefaultInterface: IFeature;
begin
  Result := FServer.DefaultInterface;
end;

{$ENDIF}

procedure Register;
begin
  RegisterComponents('Data Controls',[TPLC, TPLCConnection]);
end;

end.
```

S7UP_Interfaces.pas

```
unit S7UP_Interfaces;

{$TYPEDADDRESS OFF} // Unit must be compiled without type-checked pointers.
interface

uses Windows, ActiveX, Classes, Graphics, OleServer, OleCtrls, StdVCL;

const
  S7WCUPIntLibMajorVersion = 1;
  S7WCUPIntLibMinorVersion = 0;

  LIBID_S7WCUPIntLib: TGUID = '{F1BF8DCA-AB9F-49C1-A186-834FA930718A}';

  IID_IFeatureCallback: TGUID = '{B52F123A-C819-4A0B-ABED-71B1F5C3A2C7}';
  IID_IFeature: TGUID = '{4E00B20B-F5CA-4BEE-A599-3DE2FBED32B4}';
  IID_IFeatureAccess: TGUID = '{A200D9D2-157D-4935-BA01-701B262A88D9}';
  IID_IHelpControl: TGUID = '{60F1230F-27D8-4931-8320-F97E01FC03BA}';
  IID_IHelpAccess: TGUID = '{68A0BD83-A253-42E7-B018-3A509F4D1598}';
  IID_IPLC: TGUID = '{2F8564E3-CEC5-4747-A109-F52743F80292}';

type
  IFeatureCallback = interface;
  IFeatureCallbackDisp = dispinterface;
  IFeature = interface;
  IFeatureDisp = dispinterface;
  IFeatureAccess = interface;
  IFeatureAccessDisp = dispinterface;
  IHelpControl = interface;
  IHelpControlDisp = dispinterface;
  IHelpAccess = interface;
  IHelpAccessDisp = dispinterface;
  IPLC = interface;
  IPLCDisp = dispinterface;

// *****************************************************************//
// Interface: IFeatureCallback
// *****************************************************************//
  IFeatureCallback = interface(IDispatch)
    ['{B52F123A-C819-4A0B-ABED-71B1F5C3A2C7}']
    procedure OnFeatureChanged(const FeatureName: WideString; Context:
OleVariant;
                               NotificationID: Integer; AttributeNames:
OleVariant;
                               AttributeValues: OleVariant); safecall;
    procedure OnPLCDisconnect(ErrorID: Integer); safecall;
  end;

// *****************************************************************//
// DispIntf:  IFeatureCallbackDisp
// *****************************************************************//
  IFeatureCallbackDisp = dispinterface
    ['{B52F123A-C819-4A0B-ABED-71B1F5C3A2C7}']
    procedure OnFeatureChanged(const FeatureName: WideString; Context:
OleVariant;
                               NotificationID: Integer; AttributeNames:
```

```
OleVariant;
                              AttributeValues: OleVariant); dispid 1;
    procedure OnPLCDisconnect(ErrorID: Integer); dispid 2;
  end;

// *********************************************************************//
// Interface: IFeature
// *********************************************************************//
  IFeature = interface(IDispatch)
    ['{4E00B20B-F5CA-4BEE-A599-3DE2FBED32B4}']
    procedure GetFeature(const FeatureName: WideString; var pAttributeNames:
OleVariant;
                         var pAttributeValues: OleVariant; var pErrorID:
Integer); safecall;
    procedure SetFeature(const FeatureName: WideString; AttributeNames:
OleVariant;
                         AttributeValues: OleVariant; var pErrorID: Integer);
safecall;
    procedure RegisterFeatureForChange(const pCallback: IFeatureCallback;
                                       const FeatureName: WideString; Context:
OleVariant;
                                       var pNotificationID: Integer; var
pErrorID: Integer); safecall;
    procedure UnregisterFeatureForChange(NotificationID: Integer; var pErrorID:
Integer); safecall;
    procedure RegisterForConnectionCheck(const pCallback: IFeatureCallback;
                                         var pNotificationID: Integer; var
pErrorID: Integer); safecall;
    procedure UnregisterForConnectionCheck(NotificationID: Integer; var
pErrorID: Integer); safecall;
  end;

// *********************************************************************//
// DispIntf:  IFeatureDisp
// *********************************************************************//
  IFeatureDisp = dispinterface
    ['{4E00B20B-F5CA-4BEE-A599-3DE2FBED32B4}']
    procedure GetFeature(const FeatureName: WideString; var pAttributeNames:
OleVariant;
                         var pAttributeValues: OleVariant; var pErrorID:
Integer); dispid 1;
    procedure SetFeature(const FeatureName: WideString; AttributeNames:
OleVariant;
                         AttributeValues: OleVariant; var pErrorID: Integer);
dispid 2;
    procedure RegisterFeatureForChange(const pCallback: IFeatureCallback;
                                       const FeatureName: WideString; Context:
OleVariant;
                                       var pNotificationID: Integer; var
pErrorID: Integer); dispid 3;
    procedure UnregisterFeatureForChange(NotificationID: Integer; var pErrorID:
Integer); dispid 4;
    procedure RegisterForConnectionCheck(const pCallback: IFeatureCallback;
                                         var pNotificationID: Integer; var
pErrorID: Integer); dispid 5;
    procedure UnregisterForConnectionCheck(NotificationID: Integer; var
pErrorID: Integer); dispid 6;
```

```
  end;

// *********************************************************************//
// Interface: IFeatureAccess
// *********************************************************************//
  IFeatureAccess = interface(IDispatch)
    ['{A200D9D2-157D-4935-BA01-701B262A88D9}']
    procedure SetFeatureProvider(const PLCIdentification: WideString; const
pIFeature: IFeature;
                                 FeatureNames: OleVariant; AttributeNamesArray:
OleVariant;
                                 var pCanDisplayFeatures: Integer; var
pErrorID: Integer); safecall;
    procedure ReleaseFeatureProvider(var pErrorID: Integer); safecall;
  end;

// *********************************************************************//
// DispIntf:  IFeatureAccessDisp
// *********************************************************************//
  IFeatureAccessDisp = dispinterface
    ['{A200D9D2-157D-4935-BA01-701B262A88D9}']
    procedure SetFeatureProvider(const PLCIdentification: WideString; const
pIFeature: IFeature;
                                 FeatureNames: OleVariant; AttributeNamesArray:
OleVariant;
                                 var pCanDisplayFeatures: Integer; var
pErrorID: Integer); dispid 1;
    procedure ReleaseFeatureProvider(var pErrorID: Integer); dispid 2;
  end;

// *********************************************************************//
// Interface: IHelpControl
// *********************************************************************//
  IHelpControl = interface(IDispatch)
    ['{60F1230F-27D8-4931-8320-F97E01FC03BA}']
    procedure ShowHelp(ContextID: Integer); safecall;
    function  Get_LanguageExtension: WideString; safecall;
    procedure Set_LanguageExtension(const pVal: WideString); safecall;
    property LanguageExtension: WideString read Get_LanguageExtension write
Set_LanguageExtension;
  end;

// *********************************************************************//
// DispIntf:  IHelpControlDisp
// *********************************************************************//
  IHelpControlDisp = dispinterface
    ['{60F1230F-27D8-4931-8320-F97E01FC03BA}']
    procedure ShowHelp(ContextID: Integer); dispid 1;
    property LanguageExtension: WideString dispid 2;
  end;

// *********************************************************************//
// Interface: IHelpAccess
// *********************************************************************//
  IHelpAccess = interface(IDispatch)
    ['{68A0BD83-A253-42E7-B018-3A509F4D1598}']
    procedure SetHelpControl(const pIHelpControl: IHelpControl; var pErrorID:
```

```
Integer); safecall;
    procedure ReleaseHelpControl(var pErrorID: Integer); safecall;
  end;

// *********************************************************************//
// DispIntf:  IHelpAccessDisp
// *********************************************************************//
  IHelpAccessDisp = dispinterface
    ['{68A0BD83-A253-42E7-B018-3A509F4D1598}']
    procedure SetHelpControl(const pIHelpControl: IHelpControl; var pErrorID:
Integer); dispid 1;
    procedure ReleaseHelpControl(var pErrorID: Integer); dispid 2;
  end;

// *********************************************************************//
// Interface: IPLC
// *********************************************************************//
  IPLC = interface(IDispatch)
    ['{2F8564E3-CEC5-4747-A109-F52743F80292}']
    procedure Browse(var pConnectStrings: OleVariant; var pStartInfos:
OleVariant;
                     var pErrorID: Integer); safecall;
    procedure Connect(const ConnectString: WideString; var pIFeature: IFeature;
                      var pFeatureNames: OleVariant; var pAttributeNamesArray:
OleVariant;
                      var pErrorID: Integer); safecall;
  end;

// *********************************************************************//
// DispIntf:  IPLCDisp
// *********************************************************************//
  IPLCDisp = dispinterface
    ['{2F8564E3-CEC5-4747-A109-F52743F80292}']
    procedure Browse(var pConnectStrings: OleVariant; var pStartInfos:
OleVariant;
                     var pErrorID: Integer); dispid 1;
    procedure Connect(const ConnectString: WideString; var pIFeature: IFeature;
                      var pFeatureNames: OleVariant; var pAttributeNamesArray:
OleVariant;
                      var pErrorID: Integer); dispid 2;
  end;

implementation

uses ComObj;

end.
```

Index

Response Form

Your comments and recommendations help us to improve the quality and usefulness of our publications. Please take the first available opportunity to fill out this questionnaire and return it to Siemens.

Name of Product Documentation:

_ _

Please give each of the following questions your own personal mark within a range from 1 (very good) to 5 (very poor).

☐ Do the contents meet your requirements?

☐ Is the information you need easy to find?

☐ Is the text easy to understand?

☐ Does the level of technical detail meet your requirements?

☐ Please rate the quality of the graphics and tables.

Additional comments:

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

_ _

Please check any industry that applies to you:

☐ Automotive

☐ Chemical

☐ Electrical Machinery

☐ Food

☐ Instrument and Control

☐ Non-electrical Machinery

☐ Petrochemical

☐ Pharmaceutical

☐ Plastic

☐ Pulp and Paper

☐ Textiles

☐ Transportation

☐ Other ___________________________

Mail your response to:

Siemens Energy & Automation, Inc.

ATTN: Technical Communications

One Internet Plaza

Johnson City TN USA 37604

Include this information:

From

Name: _

Job Title: _

Company Name _

Street: _

City and State: _

Country: _

Telephone: _