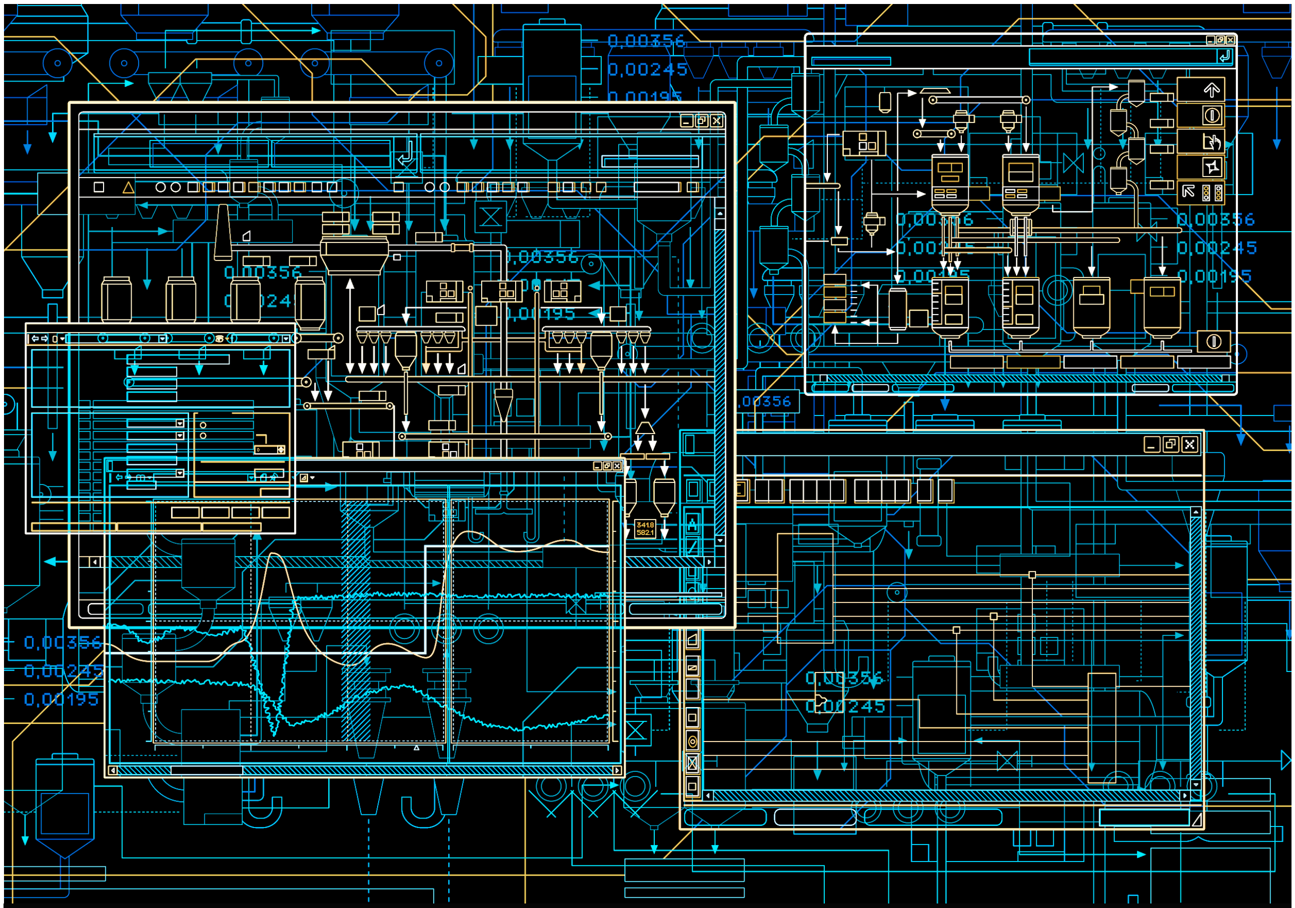


System 800xA

PLC Connect

Configuration

System Version 5.1

Power and productivity
for a better world™

System 800xA

PLC Connect Configuration

System Version 5.1

NOTICE

This document contains information about one or more ABB products and may include a description of or a reference to one or more standards that may be generally relevant to the ABB products. The presence of any such description of a standard or reference to a standard is not a representation that all of the ABB products referenced in this document support all of the features of the described or referenced standard. In order to determine the specific features supported by a particular ABB product, the reader should consult the product specifications for the particular ABB product.

ABB may have one or more patents or pending patent applications protecting the intellectual property in the ABB products described in this document.

The information in this document is subject to change without notice and should not be construed as a commitment by ABB. ABB assumes no responsibility for any errors that may appear in this document.

In no event shall ABB be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall ABB be liable for incidental or consequential damages arising from use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without written permission from ABB, and the contents thereof must not be imparted to a third party nor used for any unauthorized purpose.

The software or hardware described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license. This product meets the requirements specified in EMC Directive 2004/108/EEC and in Low Voltage Directive 2006/95/EEC.

TRADEMARKS

All rights to copyrights, registered trademarks, and trademarks reside with their respective owners.

Copyright © 2003-2010 by ABB.
All rights reserved.

Release: June 2010
Document number: 3BSE035041-510

TABLE OF CONTENTS

About This Book

General

This manual describes how to engineer and maintain PLC Connect.

Document Conventions

Microsoft Windows conventions are normally used for the standard presentation of material when entering text, key sequences, prompts, messages, menu items, screen elements, and so on.

Warning, Caution, Information, and Tip Icons

This publication includes **Warning**, **Caution**, and **Information** where appropriate to point out safety related or other important information. It also includes **Tip** to point out useful hints to the reader. The corresponding symbols should be interpreted as follows:

Electrical warning icon indicates the presence of a hazard which could result in *electrical shock.*

Warning icon indicates the presence of a hazard which could result in *personal injury.*

Caution icon indicates important information or warning related to the concept discussed in the text. It might indicate the presence of a hazard which could result in *corruption of software or damage to equipment/property.*

Information icon alerts the reader to pertinent facts and conditions.

Tip icon indicates advice on, for example, how to design your project or how to use a certain function

Although **Warning** hazards are related to personal injury, and **Caution** hazards are associated with equipment or property damage, it should be understood that operation of damaged equipment could, under certain operational conditions, result in degraded process performance leading to personal injury or death. Therefore, **fully comply** with all **Warning** and **Caution** notices.

Terminology

A complete and comprehensive list of Terms is included in the *Industrial^{IT} Extended Automation System 800xA, Engineering Concepts instruction (3BDS100972*)*. The listing included in Engineering Concepts includes terms and definitions as they that apply to the 800xA system where the usage is different from commonly accepted industry standard definitions and definitions given in standard dictionaries such as ***Webster's Dictionary of Computer Terms*.** Terms that uniquely apply to this instruction may be included here as part of this document.

Term/Acronym	Description
RTDB	Real Time Data Base. A memory resident area in which real time data collected from the process is stored in a generic format, regardless of the communication protocol used for gathering the data. Data is stored in a compact format and RTDB is characterized by very efficient procedures for read and write access.
RTU	Remote Terminal Unit

Related Documentation

A complete list of all documents applicable to the 800xA Industrial^{IT} Extended Automation System is provided in *Released User Documents (3BUA000263*)*. This document lists applicable Release Notes and User Instructions. It is provided in PDF format and is included on the Release Notes/Documentation media provided with your system. Released User Documents are updated with each release and a new file is provided that contains all user documents applicable for that release with their applicable document number. Whenever a reference to a specific instruction is made, the instruction number is included in the reference.

Section 1 Introduction

Welcome to PLC Connect

PLC Connect is a connectivity option to Industrial IT 800xA that makes it possible to connect and integrate any type of remote or local installed PLC, RTU or other type of device. The information from the connected unit can be treated in the same way as information from other 800xA System connectivities.

PLC Connect includes the following:

- Object-oriented PLC Server with alarm detection for boolean, integer and real values, data processing and scaling functions. Object types include composite process object types and extended signal types.
- Basic integration of connected PLCs and RTUs.
- Built in protocols for Comli, SattBus, Modbus Serial and Modbus TCP/IP.
- Dialed communication (Option) for Comli and Modbus Serial.
- OPC DA (1.0 and 2.05a) client functionality.
- Upload of OPC server configuration.
- Configuration data stored in Aspect Directory.
- Support for Bulk Data Manager.
- Graphic elements, such as process object and status icons.
- Faceplate examples to match the graphic elements.
- Support for multiple control networks, that is, multiple connectivity servers in separate PCs in one system.
- Support for redundant connectivity servers with PLC/OPC Server communication surveillance.

- Handling of calculated softpoints.
- Open interfaces for calculations, and other client applications.
- IEC 60870-5 protocol driver (Option).

Release Information

Read the Release Notes for information concerning the specific release of PLC Connect.

Section 2 Getting Started

Introduction

This section describes how to start engineering your system, that is, how to use PLC Connect specific process objects and signals to create representations of real world objects.

- Creating Process Object Types and Signal Types on page 16 exemplifies how to create types to base the objects and signals on.
- Creating Instances of Process Objects and Signals on page 24 exemplifies how to create objects and signals based on the created types.
- Configuring Instances of Process Objects and Signals on page 33 exemplifies how process objects and signals inherits settings from process object types and signal types.

These are only examples intended to get you started. They do not cover all configuration settings. For more information, including relevant considerations, refer to the PLC Connect online help where the configurations aspects are described in detail.

Control Network Setup

Before you can start engineering your system, a control network setup have to be done. It includes creating a Generic Control Network object for each control network in the system, and configuring the objects appropriately.

This setup is part of the post installation setup of PLC Connect, and is described in the manual *Industrial^{IT} 800xA - System Post Installation (3BUA000156*)*. At this stage, the control network setup should already have been done. In the rest of this section, it is assumed that you already have made a control network setup.

Process Objects and Signals

A process object is a representation of a real world object, such as a tank, a pump or a valve. For example, a tank object can represent a tank, while a motor object can represent a motor.

A signal is a representation of a real world process signal, such as the level in a tank, the speed of a pump or whether a valve is opened or not. For example, a volume signal can represent the volume in a tank, while a speed signal can represent the speed of a motor.

A signal can also represent an internal state, for example, when used in calculations.

Process Object Types and Signal Types

Usually, some process objects or process signals are similar to each other. For example, you can use five pump objects to represent five real world pumps of the same type. Since the process objects are similar, it is not necessary to create each process object from scratch. Instead, you first create a pump object type, and then base each pump object on that type. The pump object type is a template with inheritance behavior.

The same way as process objects are based on process object types, signals are based on signal types.

Creating Process Object Types and Signal Types

This configuration example shows how to create some extended signal types and use them in composite process object types.

There are also basic process object types, and basic signal types. However, they only exist for legacy reasons. If you engineer a new system, and do not have to import process objects and signals from previous software versions, it is recommended to only use extended signal types and composite process object types.

These basic types are not described further. Refer to the online help for more information on them.

PLC Branch Object

The PLC object (PLC branch object) in the Object Type Structure, see Figure 1, is main branch for process object types and signal types.

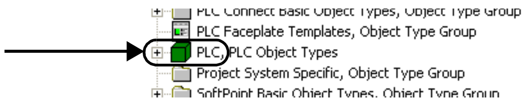

PLC Connect Basic Object Types, Object Type Group
PLC Faceplate Templates, Object Type Group
PLC, PLC Object Types
Project System Specific, Object Type Group
SoftPoint Basic Object Types, Object Type Group

Figure 1. PLC Branch Object

Configuration Example (Started)

1. In Plant Explorer, select the Object Type Structure.
2. Select the PLC branch object.
3. Select the Generic Control Network Configuration aspect. You use this aspect, see also Figure 2, when you want to create new process object or signal types.

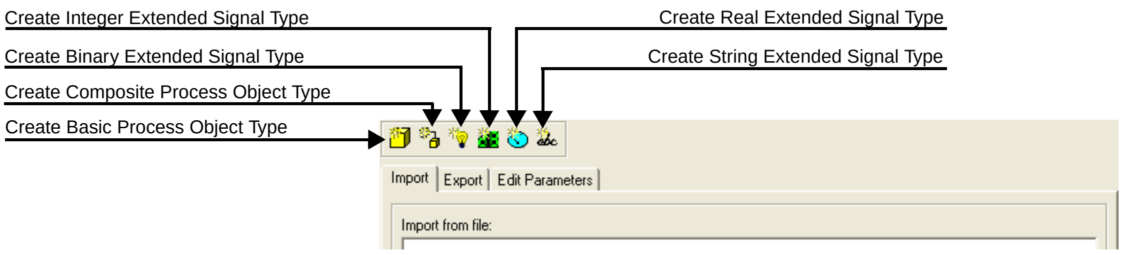

Create Integer Extended Signal Type
Create Binary Extended Signal Type
Create Composite Process Object Type
Create Basic Process Object Type
Create Real Extended Signal Type
Create String Extended Signal Type
Import
Export
Edit Parameters
Import from file:

Figure 2. Generic Control Network Configuration Aspect

Object Type Grouping

The Objects of Object Type Group can be created as child objects of the PLC object, in the Object Type Structure. These objects can be used as parents of object types derived from PLC Process Object Type, PLC Composite Process Object Type, PLC Binary Extended Signal Type, PLC Integer Extended Signal Type, PLC Real Extended Signal Type and PLC String Extended Signal Type.

Creating Object Type Group

1. Right click the PLC object and select **New Object** from the context menu.
2. Select Object Type Group from the list, enter a name and click **Create**.
3. Right click the Object Type Group object and select **New Object** from the context menu to create a new object type or signal type.

Create Signal Type

An extended signal type represents a class of signals in the real world. There are different categories of extended signal types: binary, integer, real and string.

For a specific project, you are expected to create suitable signal types, such as “TemperatureType” used in this example. Other examples of project specific signal types can be:

- A boolean output signal, named "Start", used to start a motor object.
- A boolean signal, named "AlarmNoAck", configured as an alarm with a set of specific rules for acknowledgement.
- An integer signal, named "Level", used to represent the level in a tank. The signal can have a set of pre-defined limiter alarms for high and low levels.

Signal Types Used for OPC Server Data Uploads

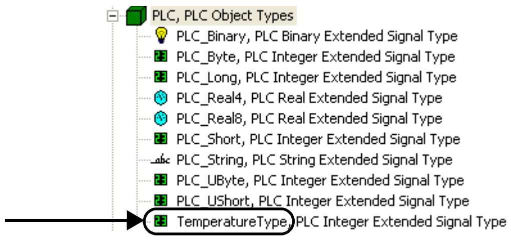
The PLC_xxx signal types, for example PLC_Binary, are pre-defined in PLC Connect. Do not change any configuration with respect to these signal types. They are used for mapping OPC server items to PLC Connect signals during an OPC server data upload. The PLC Uploader aspect is used to retrieve and filter the configuration from a connected OPC server, and automatically create PLC Connect process objects connected to the OPC server items.

For more information on the PLC Uploader aspect, refer to the online help.

Configuration Example (Continued)

4. In the Generic Control Network Configuration aspect, click the button to create a new integer signal type.

5. In the displayed dialog box, enter the signal type name. In this example, 'TemperatureType' is used.
6. Click **OK**. The new type is created as part of the PLC branch object.
7. Expand the PLC branch object to view the created signal type, see also Figure 3.



PLC, PLC Object Types
PLC_Binary, PLC Binary Extended Signal Type
PLC_Byte, PLC Integer Extended Signal Type
PLC_Long, PLC Integer Extended Signal Type
PLC_Real4, PLC Real Extended Signal Type
PLC_Real8, PLC Real Extended Signal Type
PLC_Short, PLC Integer Extended Signal Type
PLC_String, PLC String Extended Signal Type
PLC_UByte, PLC Integer Extended Signal Type
PLC_UShort, PLC Integer Extended Signal Type
TemperatureType, PLC Integer Extended Signal Type

Figure 3. Created Integer Signal Type

8. In the Generic Control Network Configuration aspect, click the button and create a new real signal type named 'VolumeType'.

Create Process Object Type

A composite process object type represents a class of related process objects and signals in the real world.

Configuration Example (Continued)

9. In the Generic Control Network Configuration aspect, click the button to create a new composite process object type.
10. In the displayed dialog box, enter the type name. In this example, 'TankType' is used.

11. Click **OK**. The new type is created as part of the PLC branch object, see also Figure 4.

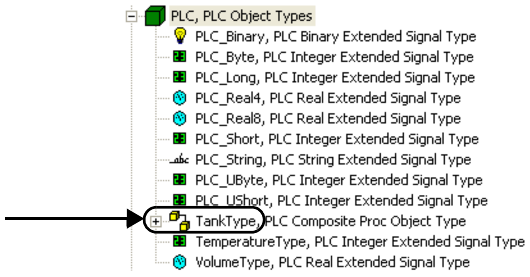

PLC, PLC Object Types
PLC_Binary, PLC Binary Extended Signal Type
PLC_Byte, PLC Integer Extended Signal Type
PLC_Long, PLC Integer Extended Signal Type
PLC_Real4, PLC Real Extended Signal Type
PLC_Real8, PLC Real Extended Signal Type
PLC_Short, PLC Integer Extended Signal Type
PLC_String, PLC String Extended Signal Type
PLC_UByte, PLC Integer Extended Signal Type
PLC_UShort, PLC Integer Extended Signal Type
TankType, PLC Composite Proc Object Type
TemperatureType, PLC Integer Extended Signal Type
VolumeType, PLC Real Extended Signal Type

Figure 4. Created Composite Process Object Type

12. Select the created object type.
13. Select the Process Object Configuration aspect. You use this aspect, see also Figure 5, when you want to add instances of other signal or object types to the created composite process object type.

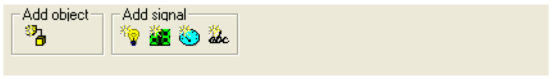
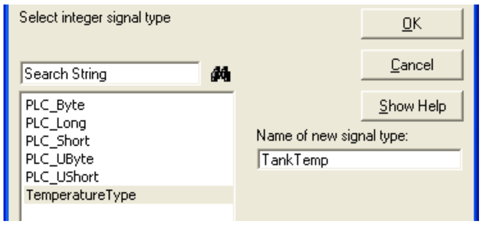


Figure 5. Process Object Configuration Aspect

Add Signal Type Instance

14. In the Process Object Configuration aspect, click the button to add an instance of an integer signal type.
15. In the displayed dialog box, select the signal type. In this example, 'TemperatureType' is selected.

16. Enter an instance name. In this example, ‘TankTemp’ is used, see Figure 6.



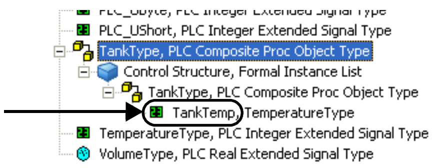
PLC_Byte
PLC_Long
PLC_Short
PLC_UByte
PLC_UShort
TemperatureType

Name of new signal type: TankTemp

Figure 6. Add Integer Signal Type Instance

17. Click **OK**. An instance of the signal type is added to the composite process object type.

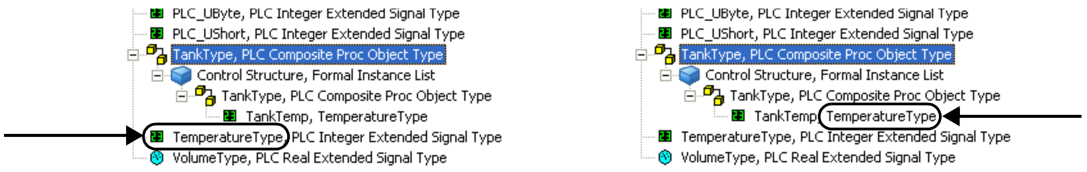
18. Expand the composite process object type to view the added instance, see also Figure 7.



PLC_UByte, PLC Integer Extended Signal Type
PLC_UShort, PLC Integer Extended Signal Type
TankType, PLC Composite Proc Object Type
Control Structure, Formal Instance List
TankType, PLC Composite Proc Object Type
TankTemp, TemperatureType
TemperatureType, PLC Integer Extended Signal Type
VolumeType, PLC Real Extended Signal Type

Figure 7. Added Signal Type Instance

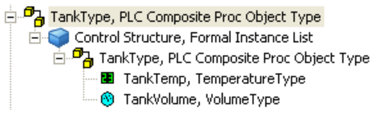
19. The last part of the instance name shows its type, see also Figure 8.



PLC_UByte, PLC Integer Extended Signal Type
PLC_UShort, PLC Integer Extended Signal Type
TankType, PLC Composite Proc Object Type
Control Structure, Formal Instance List
TankType, PLC Composite Proc Object Type
TankTemp, TemperatureType
TemperatureType, PLC Integer Extended Signal Type
VolumeType, PLC Real Extended Signal Type

Figure 8. Signal Type (Left), and Reference (Right)

20. In the Process Object Configuration aspect, click the button and add an instance of the real signal type ‘VolumeType’. Name the instance ‘TankVolume’. Figure 9 shows the end result.



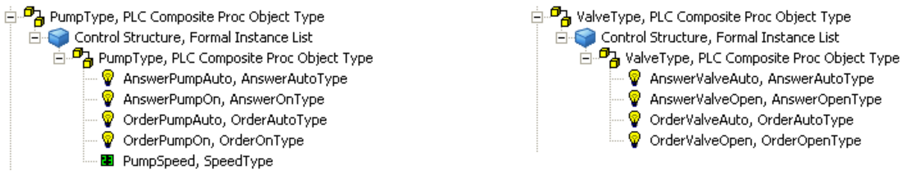
TankType, PLC Composite Proc Object Type
Control Structure, Formal Instance List
TankType, PLC Composite Proc Object Type
TankTemp, TemperatureType
TankVolume, VolumeType

Figure 9. Created Tank Object Type

Create Pump and Valve Object Types

21. Similarly, create a pump object type and a valve object type.
    a. Create two composite process object types, and relevant signal types.
    b. Add instances of the created signal types to the composite process object types.

    Figure 10 shows the object types used in this example.



PumpType, PLC Composite Proc Object Type
Control Structure, Formal Instance List
PumpType, PLC Composite Proc Object Type
AnswerPumpAuto, AnswerAutoType
AnswerPumpOn, AnswerOnType
OrderPumpAuto, OrderAutoType
OrderPumpOn, OrderOnType
PumpSpeed, SpeedType

ValveType, PLC Composite Proc Object Type
Control Structure, Formal Instance List
ValveType, PLC Composite Proc Object Type
AnswerValveAuto, AnswerAutoType
AnswerValveOpen, AnswerOpenType
OrderValveAuto, OrderAutoType
OrderValveOpen, OrderOpenType

Figure 10. Example Pump Object Type (Left) and Valve Object Type (Right)

Add Object Type Instance

22. Select the ‘TankType’ object type, and then its Process Object Configuration aspect, see also Figure 5 on page 20.
23. In the aspect, click the button to add an instance of another composite process object type.
24. In the displayed dialog box, select one of the other composite process object types. In this example, ‘PumpType’ is selected.

25. Enter an instance name. In this example, ‘TankPump’ is used, see Figure 11.

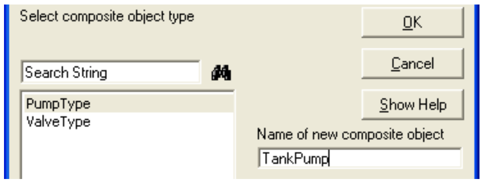

Select composite object type
Search String
PumpType
ValveType
Name of new composite object
TankPump

Figure 11. Add Object Type Instance

26. Click **OK**. An instance of the ‘PumpType’ object type is added to the ‘TankType’ object type.

27. Expand the ‘TankType’ object type to view the added instance, see also Figure 12.

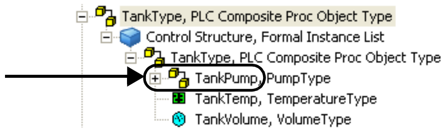

TankType, PLC Composite Proc Object Type
Control Structure, Formal Instance List
TankType, PLC Composite Proc Object Type
TankPump, PumpType
TankTemp, TemperatureType
TankVolume, VolumeType

Figure 12. Added Object Type Instance

28. In the Process Object Configuration aspect, again click the button and add an instance of the ‘ValveType’ object type. Name the instance ‘TankValve’. Figure 13 shows the result.

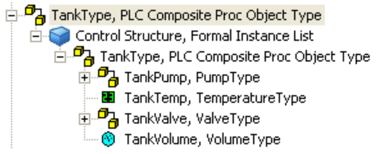

TankType, PLC Composite Proc Object Type
Control Structure, Formal Instance List
TankType, PLC Composite Proc Object Type
TankPump, PumpType
TankTemp, TemperatureType
TankValve, ValveType
TankVolume, VolumeType

Figure 13. Added Pump and Valve Object Type Instances

29. Expand the added object instances to view the finished ‘TankType’ object type, see Figure 14.

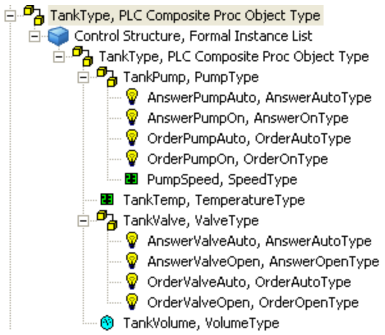

TankType, PLC Composite Proc Object Type
Control Structure, Formal Instance List
TankType, PLC Composite Proc Object Type
TankPump, PumpType
AnswerPumpAuto, AnswerAutoType
AnswerPumpOn, AnswerOnType
OrderPumpAuto, OrderAutoType
OrderPumpOn, OrderOnType
PumpSpeed, SpeedType
TankTemp, TemperatureType
TankValve, ValveType
AnswerValveAuto, AnswerAutoType
AnswerValveOpen, AnswerOpenType
OrderValveAuto, OrderAutoType
OrderValveOpen, OrderOpenType
TankVolume, VolumeType

Figure 14. Finished ‘TankType’ Object Type

Creating Instances of Process Objects and Signals

This configuration example shows how to create process objects and signals. They are based on the process object types and signal types from the previous example.

Generic Control Network Object

A Generic Control Network object represents a control network in the system, see see Figure 15 for an example. Generic Control Network objects are defined in the Control Structure.

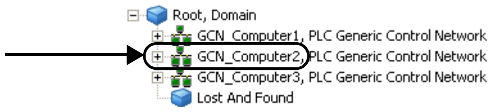

Root, Domain
GCN_Computer1, PLC Generic Control Network
GCN_Computer2, PLC Generic Control Network
GCN_Computer3, PLC Generic Control Network
Lost And Found

Figure 15. Generic Control Network Object

Configuration Example (Started)

1. In Plant Explorer, select the Control Structure.
2. Select the relevant Generic Control Network object.

3. Select the Generic Control Network Configuration aspect, see also Figure 16. You use this aspect when you want to create new controller objects, process objects or signals.

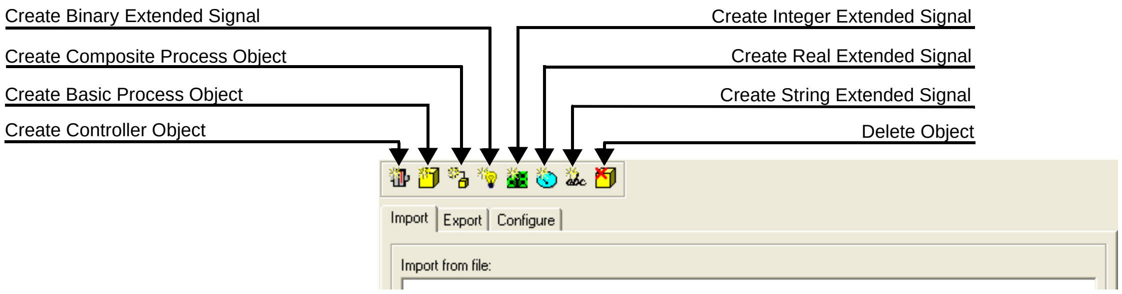

Create Binary Extended Signal
Create Composite Process Object
Create Basic Process Object
Create Controller Object
Create Integer Extended Signal
Create Real Extended Signal
Create String Extended Signal
Delete Object
Import
Export
Configure
Import from file:

Figure 16. Generic Control Network Configuration Aspect

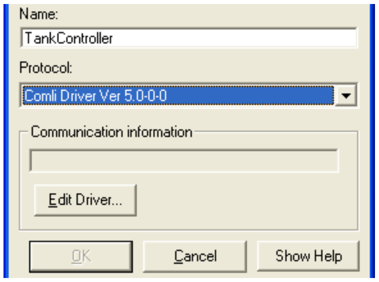
Create Controller Object

A controller object represent a controller containing process signals.

Configuration Example (Continued)

4. In the Generic Control Network Configuration aspect, click the button to create a new controller.
5. In the displayed dialog box, enter the controller name. In this example, ‘TankController’ is used.

6. Select the protocol to use for communication with the controller, see also Figure 17.



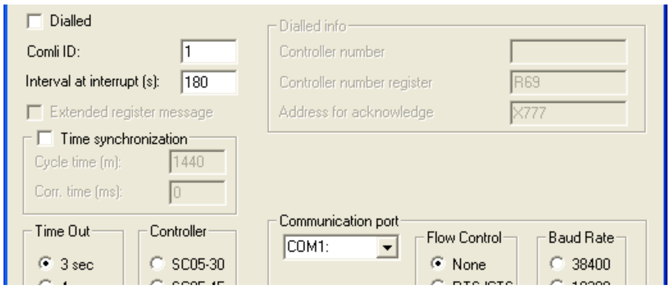
Name: TankController

Protocol: Comli Driver Ver 5.0-0-0

Communication information

Figure 17. Selected Communication Protocol

7. Click **Edit Driver**.
8. In the displayed dialog box, you can edit the driver configuration for the selected communication protocol, see Figure 18.



☐ Dialled

Comli ID: 1

Interval at interrupt (s): 180

☐ Extended register message

☐ Time synchronization

Cycle time (m): 1440

Corr. time (ms): 0

Dialled info

Controller number:

Controller number register: R69

Address for acknowledge: X777

Time Out: ● 3 sec

Controller: ○ SC05-30

Communication port: COM1:

Flow Control: ● None

Baud Rate: ○ 38400

Figure 18. Communication Parameters for Comli

Dialog appearance depends on selected communication protocol. For more information on the parameters, refer to the PLC Connect online help.

9. In this example, default settings are used. Click **OK**.
10. Return to the previous dialog box, and click **OK**. The new controller object is created.

11. Expand the Generic Control Network object to view the created controller object, also Figure 19.

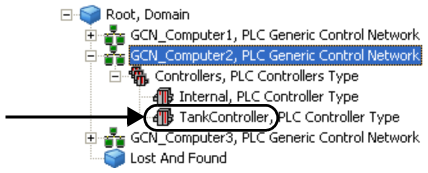

Root, Domain
GCN_Computer1, PLC Generic Control Network
GCN_Computer2, PLC Generic Control Network
Controllers, PLC Controllers Type
Internal, PLC Controller Type
TankController, PLC Controller Type
GCN_Computer3, PLC Generic Control Network
Lost And Found

Figure 19. Created Controller Object

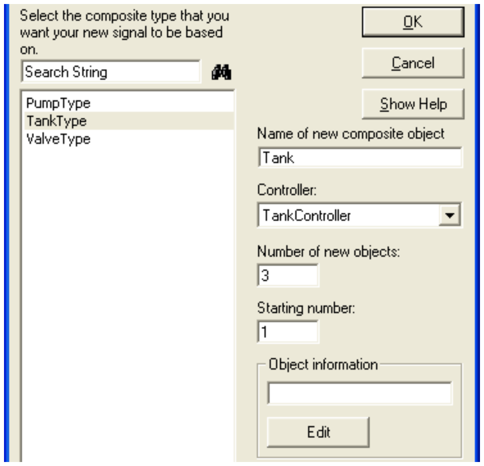
Create Process Object

A composite process object represents a group of related process objects and signals in the real world.

Configuration Example (Continued)

12. Return to the Generic Control Network Object and its Control Network Configuration aspect, see also Figure 16 on page 25.
13. Click the button to create a new composite process object.
14. In the displayed dialog box, select the relevant process object type. In this example, ‘TankType’ is selected.
15. Enter a process object name. In this example, ‘Tank’ is used.
16. Select the controller containing the relevant process signal. In this example, ‘TankController’ is selected.

17. Enter the number of new process objects to create, see also Figure 20.



Select the composite type that you want your new signal to be based on.

Search String

PumpType
TankType
ValveType

Name of new composite object: Tank

Controller: TankController

Number of new objects: 3

Starting number: 1

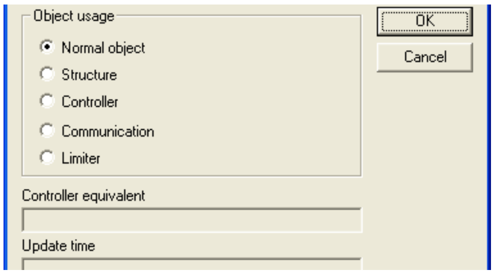
Object information

Figure 20. Add Process Objects

When more than one object is created, the character '1' will be appended to the first object name, '2' to the second and so on. If you want the numbering to start with something else than '1', enter the number in the Starting number box.

18. Click **Edit** to set process object usage.

19. In the displayed dialog box, you can set process object usage, see also Figure 21. Possible settings depend on the used communication protocol.



Object usage

[✓ Normal object] Structure Controller Communication Limiter

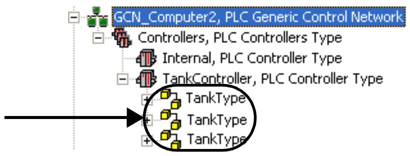
Controller equivalent

Update time

Figure 21. Object Information Settings for Comli

For more information on the settings, refer to the PLC Connect online help.

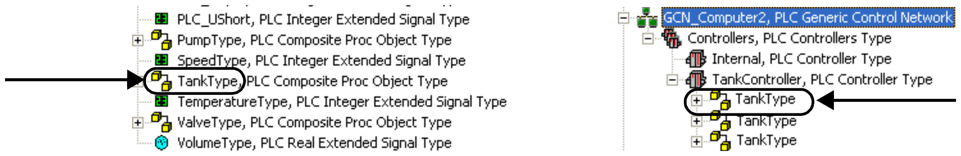
20. In this example, default settings are used. Click **OK**.
21. In the previously displayed dialog box, click **OK**. The new process objects are created as part of the selected controller.
22. Expand the controller object to view the created process objects, see also Figure 22.



```
GCN_Computer2, PLC Generic Control Network
  Controllers, PLC Controllers Type
    Internal, PLC Controller Type
    TankController, PLC Controller Type
      TankType
      TankType
      TankType
```

Figure 22. Created Process Objects

23. The last part of the process object name shows its type, see also Figure 23.



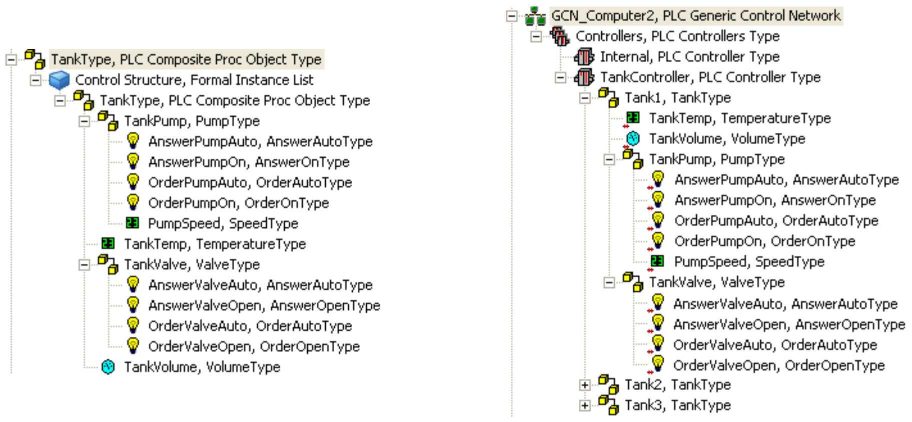
```
PLC_UShort, PLC Integer Extended Signal Type
PumpType, PLC Composite Proc Object Type
SpeedType, PLC Integer Extended Signal Type
TankType, PLC Composite Proc Object Type
TemperatureType, PLC Integer Extended Signal Type
ValveType, PLC Composite Proc Object Type
VolumeType, PLC Real Extended Signal Type
```

```
GCN_Computer2, PLC Generic Control Network
  Controllers, PLC Controllers Type
    Internal, PLC Controller Type
    TankController, PLC Controller Type
      TankType
      TankType
      TankType
```

Figure 23. Object Type Structure (Left), and Control Structure (Right)

24. Each composite process object will contain the same set of process signals and other process objects as the composite process object type. Expand an object to

view them, see also the right part of Figure 24. To compare with the object type, see also the left part of the figure.



```
TankType, PLC Composite Proc Object Type
  Control Structure, Formal Instance List
    TankType, PLC Composite Proc Object Type
      TankPump, PumpType
        AnswerPumpAuto, AnswerAutoType
        AnswerPumpOn, AnswerOnType
        OrderPumpAuto, OrderAutoType
        OrderPumpOn, OrderOnType
        PumpSpeed, SpeedType
      TankTemp, TemperatureType
      TankValve, ValveType
        AnswerValveAuto, AnswerAutoType
        AnswerValveOpen, AnswerOpenType
        OrderValveAuto, OrderAutoType
        OrderValveOpen, OrderOpenType
      TankVolume, VolumeType
```

```
GCN_Computer2, PLC Generic Control Network
  Controllers, PLC Controllers Type
    Internal, PLC Controller Type
    TankController, PLC Controller Type
      Tank1, TankType
        TankTemp, TemperatureType
        TankVolume, VolumeType
        TankPump, PumpType
          AnswerPumpAuto, AnswerAutoType
          AnswerPumpOn, AnswerOnType
          OrderPumpAuto, OrderAutoType
          OrderPumpOn, OrderOnType
          PumpSpeed, SpeedType
        TankValve, ValveType
          AnswerValveAuto, AnswerAutoType
          AnswerValveOpen, AnswerOpenType
          OrderValveAuto, OrderAutoType
          OrderValveOpen, OrderOpenType
      Tank2, TankType
      Tank3, TankType
```

Figure 24. Tank Object Type (Left), and Tank Object (Right)

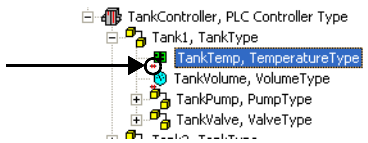
Connect Process Signal

An extended signal represents a signal in the real world, or an internal state. The former is connected externally to a hardware address in a controller, while the latter is not.

Configuration Example (Continued)

25. Select a process signal. In this example, the ‘TankTemp’ signal part of the ‘Tank1’ object is selected.

26. The icon indicates that the process signal is not connected externally, see also Figure 25.



```
TankController, PLC Controller Type
  Tank1, TankType
    TankTemp, TemperatureType
    TankVolume, VolumeType
    TankPump, PumpType
    TankValve, ValveType
```

Figure 25. Icon Indicating a Not Connected Process Signal

27. Select the Signal Configuration aspect, see also Figure 26. Under the ID tab, you can connect the signal to a hardware address in the controller.

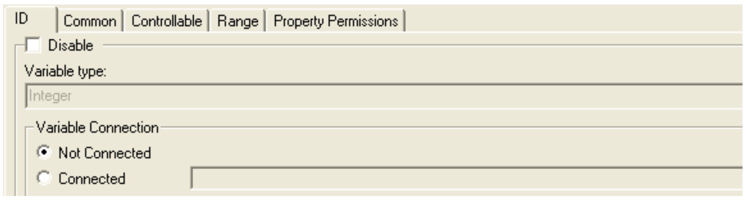


Figure 26. Signal Configuration Aspect

28. Select Connected, and then enter the relevant signal hardware address, see also Figure 27. The address format depends on the used communication protocol.

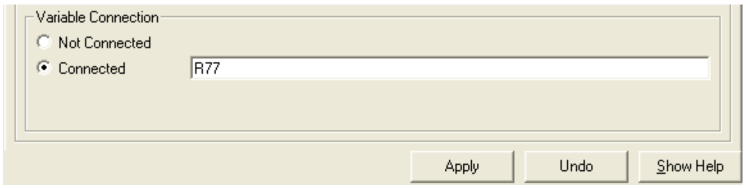


Figure 27. Entered Signal Hardware Address

For more information on valid addresses, refer to the PLC Connect online help.

If OPC communication is used, a Browse button is available. It can be used to browse for the signal.

29. Click **Apply**.

30. Similarly, connect all other process signals in the created tank objects.

Deploy

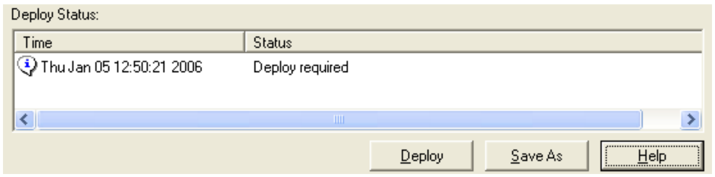
To make configuration changes take effect, you have to deploy them.

Configuration Example (Continued)

31. Select the Generic Control Network object, see also Figure 15 on page 24.

    The Deploy aspect can optionally be placed on process objects and controller objects.

32. Select the Deploy aspect, see also Figure 28.



Deploy Status:

Time	Status
Thu Jan 05 12:50:21 2006	Deploy required

Figure 28. Deploy Aspect

33. Click **Deploy**. The new configuration will take affect.

Libraries

Object types that are derived from PLC Process Object Type, PLC Composite Process Object Type, PLC Binary Extended Signal Type, PLC Integer Extended Signal Type, PLC Real Extended Signal Type and PLC String Extended Signal Type can be created as child object of a library version object. The library version needs to be placed under the PLC object in the Object Type Structure, else these objects will not be available as base types when creating instances in the Control Structure.

Using libraries with PLC Connect

1. Select the Libraries object in the Library Structure.
2. Enter the name of the library and select the PLC branch object in the **Object Type Structure insertion point for the library version**. A new library object is created in Object Type Structure.

3. Select the PLC branch object in Object Type Structure. The new library object is inserted in Object Type Structure.
4. Right click on the library version object and select **New Object** from the context menu to create a new object type or signal type.

Subfolders and Libraries for PLC Object Types

This new functionality allows PLC Connect object types to be placed in libraries or object type groups in the Plant Explorer. This facilitates creating groups of object types for easier navigation.

To create an object type group:

1. In the Structure Selector of Plant Explorer, select Object Type Structure.
2. Right-click desired PLC object and select Create Object. The New Object window appears.
3. Under the Common tab select Object Type Group and enter the object type group's name.
4. The required object type group object is created.
5. Add object types under this group.

To create a library, go to the Library Structure of Plant Explorer and right-click on Libraries to create a new object, that is, library. Select PLC object as the insertion point of the new library. The new library can then be used to create object types in the Object Type Structure.

Configuring Instances of Process Objects and Signals

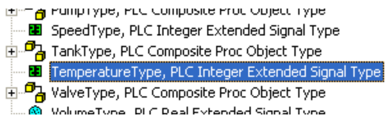
You use the PLC Connect specific configuration aspects to configure process objects and signals as desired. For information on available aspects and their settings refer to the PLC Connect online help.

However, it is important to understand the difference between default and individual settings. This configuration example shows how to change the range for a process signal from the previous example.

Configure Default Settings

Configuration settings for a signal type will be the default settings for signals based on that type. For example, if you set the range of a signal type to -50–120, then the default range for the signal will also be -50–120 due to inheritance.

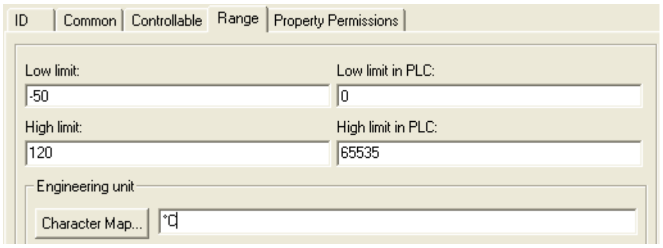
1. Select the Object Type Structure.
2. Select the PLC branch object, and then a signal type. In this example, 'TemperatureType' is selected, see also Figure 29.



SpeedType, PLC Integer Extended Signal Type
TankType, PLC Composite Proc Object Type
TemperatureType, PLC Integer Extended Signal Type
ValveType, PLC Composite Proc Object Type

Figure 29. Selected Signal Type

3. Select the Signal Configuration aspect, and the Range tab.
4. Enter the signal value range in the system to be used. In this example, -50 and 120 are entered in the Low and High limit boxes. (This range is used to scale values from the controller to the signal value range used in the system.)
5. Enter the signal value range in the controller in the Low limit in PLC and High limit in PLC boxes. In this example, the default values are used.
6. Enter an engineering unit, see also Figure 30. (You can use **Character Map** to add special characters.)



Low limit: -50
Low limit in PLC: 0
High limit: 120
High limit in PLC: 65535
Engineering unit: °C

Figure 30. Range Tab for a Signal Type

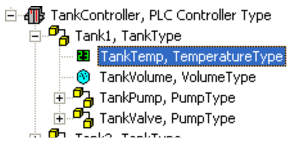
7. Click **Apply**.

Configure Individual Settings

For signals, you can select to use the default settings, or enter new individual settings. For example, if the default range for a signal is -50–120, then you can set the new range to 0–100.

8. Select the Control Structure.

9. Select relevant Generic Control Network object, and then a signal based on the signal type. In this example, 'TankTemp' is selected, see also Figure 31.

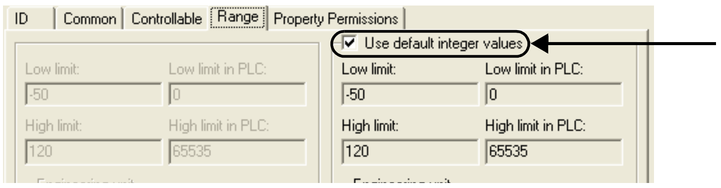


TankController, PLC Controller Type
Tank1, TankType
TankTemp, TemperatureType
TankVolume, VolumeType
TankPump, PumpType
TankValve, PumpType

Figure 31. Selected Signal

10. Select the Signal Configuration aspect, and the Range tab.

    When Use default values is selected, the default values from the signal types will be used, see Figure 32.



ID | Common | Controllable | Range | Property Permissions

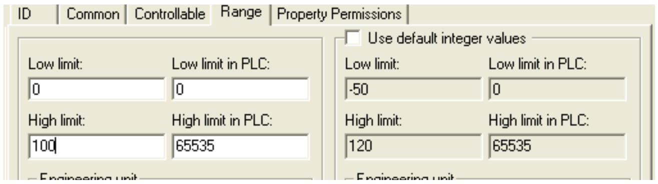
Low limit: -50
Low limit in PLC: 0
High limit: 120
High limit in PLC: 65535

[✓] Use default integer values

Low limit: -50
Low limit in PLC: 0
High limit: 120
High limit in PLC: 65535

Figure 32. Range Tab for a Signal

11. Clear the check box to use individual settings, and then enter the new values. In this example, 0 and 100 are entered as new low and high limits, see also Figure 33.


ID | Common | Controllable | Range | Property Permissions

Low limit: 0
Low limit in PLC: 0
High limit: 100
High limit in PLC: 65535

☐ Use default integer values
Low limit: -50
Low limit in PLC: 0
High limit: 120
High limit in PLC: 65535

Figure 33. Individual Settings

12. Click **Apply**.
13. The ▫ icon indicates that the process signal now has individual signal settings, see also Figure 34.

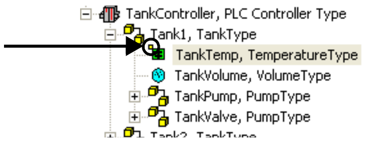

TankController, PLC Controller Type
Tank1, TankType
TankTemp, TemperatureType
TankVolume, VolumeType
TankPump, PumpType
TankValve, PumpType

Figure 34. Icon Indicating Individual Signal Settings

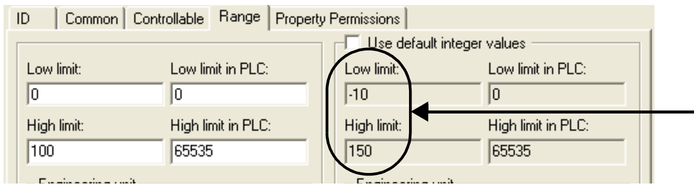
Revert to Default Settings

You can always revert back to default settings. For example, if you have set the range for a signal to 0–100 and you select to use the default settings, then the range will be -50–120 as before.

14. Select Use default values.
15. Click **Apply**. The settings will revert to default values, see also Figure 32.

Change Default Settings

When you change the settings for a signal type, this will affect all signals based on that type. For example, if you change the range of a signal type from -50–120 to -10–150, then the default range for the signal will also be changed from -50–120 to -10–150. However, individual settings are not affected by the changed defaults, see also Figure 35.



Low limit: 0
Low limit in PLC: 0
High limit: 100
High limit in PLC: 65535

☐ Use default integer values
Low limit: -10
Low limit in PLC: 0
High limit: 150
High limit in PLC: 65535

Figure 35. Changed Default Settings

Another Inheritance Example

The table below describes the behavior when settings are changed.

Table 1. Changed Settings

Changed Settings	Signal Type Range (Object Type Structure)	Signal Range (Control Structure)
1. Using default signal settings (in Control Structure)	0–100	0–100
2. Changing signal type settings (in Object Type Structure)	0–200	0–200
3. Entering individual signal settings (in Control Structure)	0–200	0–500
4. Changing signal type settings (in Object Type Structure)	0–300	0–500
5. Reverting to default signal settings (in Control Structure)	0–300	0–300

Section 3 Alarm and Event List Configurations

PLC Connect supplies two customized alarm and event list configurations. By associating them with Alarm and Event List aspects, you can create PLC Connect specific alarm and event lists configured to show a selection of the alarms and events.

- List of Control Blocked Signals. This configuration shows the manually controllable signals that are disabled. The values of a control blocked signal can't be changed by an operator.
- List of Forced Signals. This configuration shows forced signals. Forced signals are signals whose values are temporarily forced to a certain value, that is, the signal values appear to be constant regardless of the actual process values.

PLC alarm list exists only for the backward compatibility.

The customized alarm and event list configurations are located in the Library Structure under the object "Alarm & Event List Configurations", sub-object "SPS Alarm & Event List Configuration".

Alarm and event list configuration is described in the *Industrial IT 800xA - System Operator Workplace Configuration (3BSE030322*)*.

Event Configurations

Event configuration influence the appearence of the alarm and event list.

Alarm Text Group

Alarm Text Groups is used in Alarm and Event Configuration aspect and displayed in the Alarm and Event List.

Add Alarm Text Group

1. Select the **Alarm Text Groups** object in the Library Structure.
2. Add a PLCC **Alarm Text Group** object.
3. Select the Alarm Text Group configuration aspect on the object.
4. Enter values for *Disable, Enable, Autodisable, Event On, Event Off and Acknowledge* text-boxes.
5. Click the **Apply** button.

The Control, Force, Unforce, Disable Control and Enable Control texts/fields are moved to the Operator Actions Text Group.

Delete Alarm Text Group

1. Select the **Alarm Text Groups** object in the Library Structure.
2. Click **Delete the PLCC Alarm Text Group Object.**
3. A dialog “Do you wish to scan signals for the Alarm Text Group you are about to delete? This may take several minutes to complete, depending on the amount of signals to scan. If you choose No, the alarm text group will be unconditionally deleted” appears.
4. Click **Yes**. A dialog “Signals with Alarm text group displays all signals where the deleted alarm text group are used” appears.

Section 4 External Access to Process Data and Configuration

Introduction

This section describes various methods for client applications to access PLC Connect Real Time Data and other means of interaction with PLC Connect:

- Extension Processes on page 41 describes the use of extension processes in the PLC Server.
- PLC Connect Real Time Data Access on page 42 describes an interface for project specific variable access in the Real Time Database (RTDB).
- Communication Server Pre Treatment on page 60 describes an interface for project specific variable pre treatment in the Communication Server.
- PLC Connect Properties on page 67 describes accessible properties.
- Dial Manager Server Access on page 79 describes an interface for accessing the Dial Manager Server.

Extension Processes

You can use applications of your own together with PLC Connect. When you want such an application to start and stop synchronized with the PLC Server, you have to add the application as an extension process to the PLC Server. However, the application you add cannot have any kind of user interface that require manual input, for example, clicking an **OK** button. This kind of user interface will halt program execution, since extension processes (you can have more than one) run in the same service context as the PLC Server.

In a redundant PLC Connect connectivity server environment, the PLC Connect Server Service ensures that an extension process executes only in the node that is currently master. Automatic restart of the extension process in the event of a process crash is also supported. However, the PLC Connect built in mechanism for real time data replication between the two nodes, cannot be utilized. If the implementation in an extension process in a redundant connectivity server environment requires any kind of data replication in order to function, this has to be solved by the extension process itself.

To add your application as an extension process:

1. Start a Plant Explorer Workplace.
2. Select the Service Structure.
3. Select the object named “PLC Server, Service”.
4. Select the Service Group object for the desired control network.
5. Select the Service Group Definition aspect.
6. Select the Special Configuration tab.
7. Click **Help** and refer to the information on how to add the application as an extension process.
8. If you want to add the application as an extension process for other control networks, repeat Step 4 to Step 7 for each of these control networks.

PLC Connect Real Time Data Access

Introduction

The real time data in the RTDB is accessible for user written applications via a COM interface. This Open Interface (OIF) is available on both server and client nodes. Running the application on a server node can be done for example as an extension process, see Extension Processes on page 41, but consider the restrictions that apply to such a process. It can also be run as a standalone process with or without a user interface. On a client node it cannot be run as an extension process to the PLC Server, only as a standalone executable, with or without a user interface.

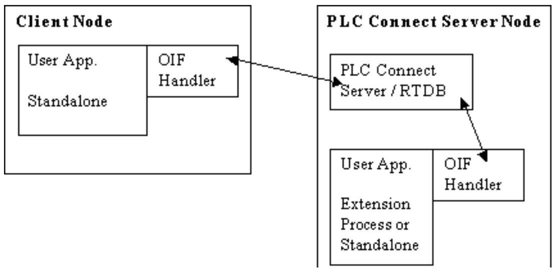


Figure 36. Client and Server Nodes

Variable Access

The variable access is performed by one or more executable files independently from the RTDB.

In the RTDB two classes are implemented: **Variables** and **Variable.**

Client means a certain instance of a Variables object, see Variable Access Interface on page 45. A program (.EXE file) can have several instances of Variables, although in most cases there is only one instance.

The RTDB is able to inform the respective EXE file when a certain variable changes its value.

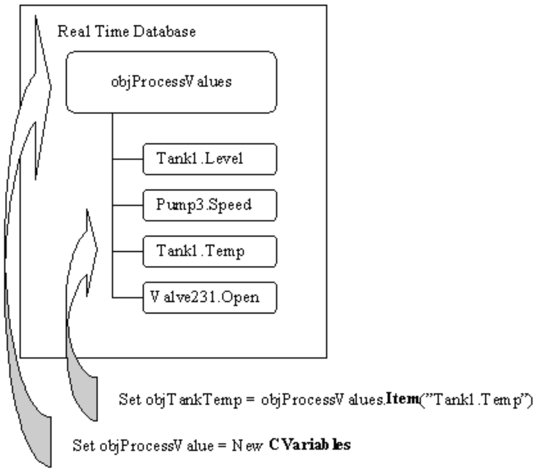

Real Time Database
objProcessValues
Tank1.Level
Pump3.Speed
Tank1.Temp
Valve231.Open
Set objTankTemp = objProcessValues.Item("Tank1.Temp")
Set objProcessValue = New CVariables

Figure 37. Variable Access

An object is created based on the class **Variables**. Using the methods for the class it is possible to create **Variable** objects or direct references to PLC Connect instances.

It is possible to Lock variables before writing values to them, via methods in the Variable Access Interface, see Variable Access Interface on page 45.

The holder of a Lock is identified by NodeId and UserId.

Locking can be done on a signal level, however the whole object is then locked.

Subscription

When a client (Variables object) is created, a queue will be allocated. In this queue variable changes will be stored. In one queue changes from all subscribed variables will be stored.

- Each client may have a queue of its own which is independent of other clients.
- It is not necessary to use the queue function, the calculation results can be written at any time, for example, cyclically.

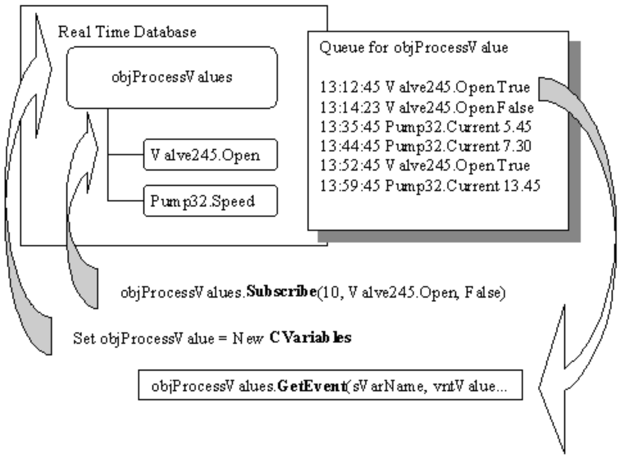

Real Time Database
objProcessValues
Valve245.Open
Pump32.Speed
Queue for objProcessValue
13:12:45 Valve245.Open True
13:14:23 Valve245.Open False
13:35:45 Pump32.Current 5.45
13:44:45 Pump32.Current 7.30
13:52:45 Valve245.Open True
13:59:45 Pump32.Current 13.45
objProcessValues.Subscribe(10, Valve245.Open, False)
Set objProcessValue = New CVariables
objProcessValues.GetEvent(sVarName, vntValue...

Figure 38. Subscription

- Any number of clients can choose not to use queues.

Variable Change

A **variable change** is defined as an analog signal value that is changing by a value higher than the defined signal hysteresis (the hysteresis may also be 0) or a binary signal change (leading, trailing or both edges).

Variable Access Interface

This section describes the properties and methods of the interface for **“Variable access”**. This interface consists of two different classes: **Variable** and **Variables**. To be able to use these classes, for example in Visual Basic, references have to be done to a library included in the PLC Connect installation. The name of the library is **AdsScadaHdlr 1.0 Type Library**.

Variable Access Properties and Methods

The properties and methods of the classes are listed in Table 2. All properties are read only, unless anything otherwise stated in the table.

Properties such as Lower can be read both via a method and via properties. Here, the properties are considered the "normal" way to read these properties. The methods should be considered a way to optimize the performance by reading several properties with the same call.

Table 2. Properties and Methods

Class	Methods/properties
Variables	Init(...)
Variables	Init2(...)
Variables	Valid As Boolean
Variables	Changed As Boolean
Variables	Subscribe (…) As Boolean
Variables	SubscribeByCalcType(…) As Boolean
Variables	GetEvent(…) As Boolean
Variables	GetQueueInfo(…) As Boolean
Variables	ClearQueueInfo()
Variables	Item(…) As Variable
Variables	ReadValue(…) As Boolean
Variables	WriteValue(…) As Long
Variables	ForceLock(…) As Long
Variables	ReleaseLock(…) As Long
Variables	LockedBy(…) As Long
Variables	GetGUID(…) As Long

Table 2. Properties and Methods (Continued)

Class	Methods/properties
Variable	ReadValue(…) As Boolean
Variable	WriteValue(…) As Long
Variable	ReadMeasure(…) As Boolean
Variable	ReadAttributes(…) As Boolean
Variable	ReadLimit(…) As Boolean
Variable	WriteLimit(…) As Long
Variable	ForceLock(…) As Long
Variable	ReleaseLock(…) As Long
Variable	LockedBy(…) As Long
Variable	Lower As Double
Variable	Upper As Double
Variable	LowerControlLim As Double (Read/Write)
Variable	UpperControlLim As Double (Read/Write)
Variable	IsInverted As Boolean
Variable	IsEvent As Boolean
Variable	IsTimeStamped As Boolean
Variable	IsBlocked As Boolean
Variable	IsControllable As Boolean
Variable	IsControlBlocked As Boolean
Variable	IsForced As Boolean
Variable	Name As String
Variable	GUID As String

In these methods the sVarName in-parameter is the name of the variable in the PLC Server. If the name is entered in the format “{xxxxxxxx-xxxx-xxxx-xxx-xxxxxxxxxxxx}”, it will be interpreted as a GUID for a variable in the PLC server.

The variable name must be entered in the format “<controller name>:<object name>. <signal name>”.

If composite process objects are used, there may be several <object name> items in SignalName. See also the code sample referred to in Sample Code on page 58.

Properties and Methods for the Class Variables

Init

Syntax: `Init(pSysCtxId As AfwSystemContextId, pSGId As AfwServiceGroupId)`

Used to identify the server.

pSysCtxId (in): the System Context Id.

pSGId (in): the Service Group Id.

Init2

Syntax: `Init2(pObject As AfwObject)`

Used to identify the server.

pObject (in): an object in the PLC Generic Control Network tree.

Valid

Syntax: `Valid As Boolean`

Always TRUE. (Exists only for backward compatibility reasons).

Changed

Syntax: `Changed As Boolean`

TRUE when there are one or several variable changes in the queue.

Subscribe

Syntax: `Subscribe(lQueueMaxSize As Long, sVarName As String,bFilter As Boolean ) As Boolean`

Subscribes a variable for queue handling. Returns TRUE if variable found.

lQueueMaxSize (in): states the maximum number of allowed events in the queue. Is only significant at the first call.
sVarName(in): returns the name of the variable.
bFilter (in): if TRUE, one variable change for each signal at most is queued (the latest).

SubscribeByCalcType

Syntax: `SubscribeByCalcType(lQueueMaxSize As Long lCalcTypeLow As Long, lCalcTypeHigh As Long, bFilter As Boolean ) As Boolean`

For future use.

GetEvent

Syntax: `GetEvent( sVarName As String, vntValue As Variant, OPCQuality As Integer, TimeStamp As Date, lCalculationType As Long, sVarParam As String, bOverFlow As Boolean ) As Boolean`

Gets a variable change from the queue. Returns FALSE if the queue is empty.

SVarName(out): returns the name of the variable.
vntValue(out): returns the value of the signal.
OPCQuality(out): indicates the OPC quality of the variable.
TimeStamp(out): time stamp for value changed.
lCalculationType(out): selected calculation type. This user defined private data is selected in the Calculations dialog box. To access it, click **Calculations** under the Common tab of a Signal Configuration aspect for an I/O signal type or signal.
sVarParam(out): entered subscription parameter. This user defined private data is also entered in the Calculations dialog box. To access it, see above.
bOverFlow (out): is TRUE if the number of events in the queue = lQueueMaxSize.

GetQueueInfo

Syntax: `GetQueueInfo(lCurrentQueueSize As Long, lPeakQueueSize As Long, lTotalQueuedEvents As Long) As Boolean`

Gives diagnostic information about a queue. Returns TRUE if call succeeded.

lCurrentQueueSize(out): returns size of queue
lPeakQueueSize(out): returns peak size value of the queue
lTotalQueuedEvents(out): returns number of queued events

ClearQueueInfo

Syntax: `ClearQueueInfo()`

Clears diagnostic information for a queue.

Item

Syntax: `Item( sVarName As String) As Variable`

Gets a reference to a named variable. If the variable is not found, ”**Nothing**” is returned.

sVarName(in): name of the variable.

ReadValue

Syntax: `ReadValue( sVarName As String, vntValue As Variant, OPCQuality As Integer ) As Boolean`

Reads the value of a named variable. FALSE if the variable is not found.

sVarName(in): name of the variable.
vntValue(out): returns the value of the signal.
OPCQuality(out): indicates the OPC quality of the variable.

There are also variants of the ReadValue method, see More on Reading and Writing on page 58.

WriteValue

If you write data to a signal connected to an external IO then that data will be written to the controller.

Syntax: `WriteValue( sVarName As String, vntValue As Variant, OPCQuality As Integer ) As Long`

Writes to a named variable, but not if **vntValue** is ”**vtEmpty**”. This is useful if you only want to write into the error bits. A variable is “**vtEmpty**” if it is declared but not assigned. If the variable is fetched from a controller, the **vntValue** is written to the controller if the variable is controllable. Otherwise, the functions connected to the signal are performed, for example the alarm function.

Returns status codes according to Table 3 on page 57.

sVarName(in): name of the variable.
vntValue (in): value to the signal.
OPCQuality (in): if the variable is of the type ”**Internal**”, then it is written to the OPC quality of the variable, else this parameter is ignored.

There are also variants of the WriteValue method, see More on Reading and Writing on page 58.

ForceLock

Syntax: `ForceLock( sVarName As String ) As Long`

Called by an application if it needs to Lock a certain variable. The Lock will be held until it is released by the application or until another application forces the Lock. If the variable is already locked by other application, this call will force the Lock.

Returns status codes according to Table 3 on page 57.

sVarName(in): name of the variable.

ReleaseLock

Syntax: `ReleaseLock( sVarName As String ) As Long`

Called by an application when it wants to release a Lock for the variable.

Returns status codes according to Table 3 on page 57.

sVarName(in): name of the variable.

LockedBy

Syntax`: LockedBy( sVarName As String, sNodeId As String, sUserId As String ) As Long`

Method that returns the identification of the user, currently holding the Lock.

Returns status codes according to Table 3 on page 57. If the variable isn't locked, nodeId and UserId will be set to 200.

sVarName(in): name of the variable.
sNodeId(out): name of the node where the Lock is held.
sUserId(out): name of the user that holds the Lock.

GetGUID

Syntax`: GetGUID( sVarName As String, sGUID As String ) As Long`

Method that returns the GUID of the variable. Returns status codes according to Table 3 on page 57. The method is intended to be used whenever an Event occurs and the client needs to know the Event's guid, since the GetEvent() function returns an object name and not a guid.

sVarName(in): name of the variable.
sGUID(out): GUID of the variable.

Properties and Methods for the Class Variable

ReadValue

It is recommended to instead use the method ReadValue in the class Variables, see also ReadValue on page 50.

Syntax`: ReadValue( vntValue As Variant, OPCQuality As Integer ) As Boolean`

Reads the value of the variable.

VntValue(out): returns the value of the signal.
OPCQuality(out): indicates the OPC quality of the variable.

WriteValue

It is recommended to instead use the method WriteValue in the class Variables, see also WriteValue on page 51.

If you write data to a signal connected to an external IO then that data will be written to the controller.

Syntax: `WriteValue( vntValue As Variant, OPCQuality As Integer ) As Long`

Writes **vntValue** to a named variable. If the variable is fetched from a controller, the value is written to the controller if the variable is controllable. Else, the functions connected to the signal, are performed, for example the alarm function. If **vntValue** contains ”**vtEmpty**”, only the quality is written, not the value. This is useful if you only want to write into the quality. A variable is “**vtEmpty**” if it is declared but not assigned.

Returns status codes according to Table 3 on page 57.

VntValue(in): value to the signal.
OPCQuality(in): indicates the OPC quality of the variable.

ReadMeasure

Syntax: `ReadMeasure( fLower As Double, fUpper As Double, fLowerControlLim As Double, fUpperControlLim As Double, bIsControllable As Boolean, bIsControlBlocked As Boolean, bIsForced As Boolean) As Boolean`

Reads measuring properties of the instance. Returns FALSE if the signal is not analog.

fLower(out): Returns the lower limit of the range of measurement.
fUpper(out): Returns the upper limit of the range of measurement.
fLowerControlLim(out): Returns the lower control limit.
fUpperControlLim(out): Returns the upper control limit.
bIsControllable(out): indicates if the signal is controllable.
bIsControlBlocked(out): indicates if the signal is control blocked.
bIsForced As Boolean(out): indicates if the signal is forced.

ReadAttributes

Syntax: `ReadAttributes(bIsInverted As Boolean, bIsEvent As Boolean, bIsTimeStamped As Boolean, bIsBlocked As Boolean, bIsControllable As Boolean, bIsControlBlocked As Boolean, bIsForced As Boolean) As Boolean`

Returns FALSE if the signal is not boolean.

bIsInverted(out): returns TRUE if the signal is inverted.
bIsEvent(out): returns TRUE if the signal is an event or an alarm.
bIsTimeStamped(out): returns TRUE if the signal is an alarm time stamped in the controller.
bIsBlocked(out): returns TRUE if the signal is blocked.
bIsControllable(out): returns TRUE if the signal is controllable.
bIsControlBlocked(out): returns TRUE if the signal is control blocked.
bIsForced(out): returns TRUE if the signal is forced.

ReadLimit

Syntax: `ReadLimit(lLimitNumber As Long,lTypeLimiter As Long, fOnLimit As Double, fOffLimit As Double) As Boolean`

Returns FALSE if the signal is not analog.

lLimitNumber(in): state the sequence number of the limiter from 1 and up to 8, for the limiters **enabled** in the object, in consecutive order from top to bottom on the limiter tab in the Configuration Tools. For more info on Configuration Tools, see the PLC Connect online help.
lTypeLimiter(out): has the value 0 for minimum-limiter and the value 1 for max.
fOnLimit(out): **fOnLimit** is the limit where the alarm is activated.
fOffLimit(out): is the limit where the alarm is deactivated.

WriteLimit

Syntax: `WriteLimit(lLimitNumber As Long,lTypeLimiter As Long, fOnLimit As Double, fOffLimit As Double) As Long`

Affects only RTDB, and not configuration values set in PLC Connect configuration aspects.

Returns status codes according to Table 3 on page 57. For example, a status code is returned if arg LimitNumber + Limittype is out of range, that is, not in [1,8] and [0,1] respectively.

lLimitNumber(in): states the sequence number of the limiter from 1 and upwards for the limiters **enabled** in the object, in consecutive order from top to bottom on the limiter tab in the Configuration Tools.
lTypeLimiter(in): has the value 0 for minimum-limiter and the value 1 for max.
fOnLimit(in): **fOnLimit** is the limit where the alarm is activated.
fOffLimit(in): is the limit where the alarm is deactivated.

ForceLock

Syntax: `ForceLock() As Long`

Called by an application if it needs to Lock a certain variable. The Lock will be held until it is released by the application or until another application forces the Lock. If the variable is already locked by other application, this call will force the Lock.

Returns status codes according to Table 3 on page 57.

ReleaseLock

Syntax: `ReleaseLock() As Long`

Called by an application when it wants to release a Lock for the variable. Returns status codes according to Table 3 on page 57.

LockedBy

Syntax: `LockedBy( sNodeId As String, sUserId As String ) As Long`

Method that returns the identification of the user, currently holding the Lock. Returns status codes according to Table 3 on page 57.

sNodeId(out): name of the node where the Lock is held.
SuserId(out): name of the user that holds the Lock.

Conditions When Writing a Variable Value

For the methods WriteValue and WriteLimit, the following is valid:

1. If the signal is blocked or forced nothing happens, but error status is returned.

2. If the signal is internal, the value is written unconditionally, no check is done on ”controllable”, ”control limits” or ”range of measurement”. A check is done, however, on control limits if the signal is controllable.
3. On an internal, real or integer signal, a check is performed on limiters.
4. On an internal binary signal, which is an event or an alarm, a possible change of state will be further event handled.
5. If the signal is external and analog it is checked that the signal is controllable and that the value is within the control limits. After this, the value is written to the controller. Possible limiters are not checked (it will happen when the new value is read the next time). If control is not performed, due to any of the conditions, error status is returned instead.
6. If the signal is external and binary it is checked that the signal is controllable. Then the value is written to the controller. If the signal is an event or an alarm, a possible change of state is not event handled (it will happen when the new value is read the next time). If control is not performed, due to any of the conditions, error status is returned instead.
7. If a control is performed to the controller, or internally at an internal signal, and event logging of operator’s actions in the database is selected, the control is event logged.

Status Codes for Methods Returning a Value of the Type Long

Some methods are returning “Long”. This could be used as a status check of the method, see Table 3.

Table 3. Returned Status Codes[1]

Value	Description	Variables								Variable				
		GetGuid	**ForceLock**	**ReleaseLock**	**LockedBy**	**WriteValue**	**WriteValueEx**	**WriteValues**	**WriteValuesEx**	**ForceLock**	**ReleaseLock**	**LockedBy**	**WriteValue**	**WriteLimit**
0	Other error, not specified below.	X	X	X	X	X	X	X	X	X	X	X	X	X
32000	Success; if the method involves writing, the value has been written. All other status codes indicate that the value is **not** written.	X	X	X	X	X	X	X	X	X	X	X	X	X
32001	The variable cannot be found in RTDB. The reason may be a misspelled name or a deploy has not been made after changes in the database.	X	X	X	X	X	X	X	X	X	X	X	X	X
32003	Variable cannot be manually controlled. The reason may be: (1) variable is not defined as controllable, (2) variable is control blocked, or (3) forced value.					X	X	X	X				X	
32004	Internal error in RTDB.	X	X	X	X	X	X	X	X	X	X	X	X	X
32009	Value is outside defined control limits.					X	X	X	X				X	
32101	Limit (number or type) is not valid.													X
32105	Variable is not reserved.				X							X		

(1) X marks methods returning the status code.

OPC Quality(In/Out)

This flag represent the quality state for a variable's data value.

OPC quality is defined as (OPC_QUALITY_MASK & opcquality) = OPC_QUALITY_GOOD(C0) or OPC_QUALITY_BAD(0x).

Sample Code

Sample Variable Access

A Visual Basic 6.0 sample, **ReadWrite.vbp**, can be found in the folder **\...\ABB Industrial IT\Operate IT\PLC Connect\Samples\PLCVariableAccess\ReadWrite**.

Sample Subscribe

A Visual Basic 6.0 sample, **Subscribe.vbp**, can be found in the folder **\...\ABB Industrial IT\Operate IT\PLC Connect\Samples\PLCVariableAccess\Subscribe**.

More on Reading and Writing

The ReadValue and WriteValue methods described in ReadValue on page 50 and WriteValue on page 51, allow you only to access one variable at a time. There are also variants of these methods, declared as shown below.

If you write data to a signal connected to an external IO then that data will be written to the controller.

Reading and Writing Time Information

The ReadValueEx and WriteValueEx methods allows you to also read and write time information to a single variable;

```
Function ReadValueEx(sVarName As String, vntValue As Variant,
pwOPCQuality As Integer, timeStamp As Date, UTCTime As
Boolean) As Boolean
```

```
Function WriteValueEx(sVarName As String, vntValue As Variant,
wOPCQuality As Integer, timeStamp As Date, UTCTime As Boolean)
As Long
```

UTCTime: whether local time (value is FALSE) or UTC time (value is TRUE) is used.

timestamp: the date and time.

WriteValueEx returns status codes according to Table 3 on page 57.

Reading and Writing Arrays of Data

The ReadValues and WriteValues methods allows you to read and write arrays of variables and their values in one call; The ReadValuesEx and WriteValuesEx methods allows you to also read and write time information.

These methods are only available for usage from C++, not Visual Basic.

```
HRESULT WriteValues([in] unsigned long count,
   [in,size_is(count)] AwfObjectId* objectIds,
   [in,size_is(count)] VARIANT* vntValues,
   [in,size_is(count)] short* wOPCQualities,
   [out,size_is(,count)] LONG** results);

HRESULT ReadValues([in] unsigned long count,
   [in,size_is(count)] AwfObjectId* objectIds,
   [out,size_is(,count)] VARIANT** vntValues,
   [out,size_is(,count)] short** wOPCQualities,
   [out,size_is(,count)] BOOL** results);

HRESULT ReadValuesEx ( [ in ] unsigned long count,
   [ in,size_is(count) ] AfwObjectId* objectIds,
   [ out,size_is(,count) ] VARIANT** vntValues,
   [ out,size_is(,count) ] short** wOPCQualities,
   [ out,size_is(,count) ] FILETIME** timeStamps,
   [ in ] VARIANT_BOOL UTCTime,
   [ out,size_is(,count) ] BOOL** results ) ;

HRESULT WriteValuesEx ( [ in ] unsigned long count,
[ in,size_is(count) ] AfwObjectId* objectIds,
[ in,size_is(count) ] VARIANT* vntValues,
[ in,size_is(count) ] short* wOPCQualities,
[ in,size_is(count) ] FILETIME* timeStamps,
[ in ] VARIANT_BOOL UTCTime,
[ out,size_is(,count) ] LONG** results);
```

UTCTime: whether local time (value is FALSE) or UTC time (value is TRUE) is used.

timestamps: the date and time.

WriteValuess and WriteValuesEx return status codes according to Table 3 on page 57.

Communication Server Pre Treatment

Introduction

The functions described in this section refer to an interface for project specific variable pre treatment in the Communication Server.

The interface is implemented essentially as a COM Interface.

Variable Pre Treatment

When a new value is received from the driver, the Communication Server calls a default COM-interface in a DLL. In the DLL, project-specific calculations can be entered. The Communication Server is the client.

The interfaces only support signals and **not** the access of total objects.

One and only one DLL can be used per server.

Variable Pre Treatment

It is presumed that the calls are made with ”In process” (that is to DLL) COM for best performance. Sample code for Visual Basic 6.0 and for Visual Studio 2005 C++ is available in the folder ...**\ABB Industrial IT\Operate IT\PLC**

Connect\Samples\AdsPreTreat\PreTreat.txt for changing the component id. To activate PreTreat4 type components ProgId must be configured in the Generic Control Network Configuration aspect at the Generic Control Network object

PreTreat3 exists only for backward compatibility.

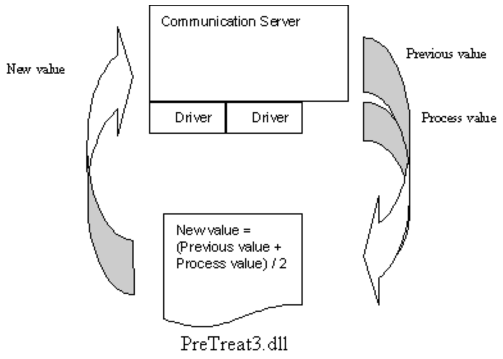


Figure 39. Variable Pre Treatment

Consider the following:

- The DLL must not implement any kind of user interface.
- The DLL must not utilize the Variable Access Interface described in PLC Connect Real Time Data Access on page 42.
- The DLL must only use the callback interface provided in the Initialize call in order to access RTDB. (See also Initialize on page 62 and Callback Interface on page 65).
- The DLL must not access any other data in the 800xA System, for example the Aspect Directory.
- The DLL must not make any file access on disk. To use log or troubleshooting files during a development phase is ok, but not during live operations.

Ensure that the code implemented is fast in execution. The DLL executes In-process in the PLC Connect Communication Server Process, and may cause the PLC Server as a whole to behave poorly due to a bad PreTreat implementation.

If a ProgId for a PreTreat4 component is configured the component is loaded by the Communication Server at program start. If a PreTreat4 component is not configured and a PreTreat3.dll is registered it is loaded by the Communication Server at program start. The component can be loaded or unloaded during a deploy operation if the PreTreat ProgId configuration is changed. By default, the PreTreat3.dll is not registered during the installation of PLC Connect. To use it, you have to register the DLL manually with the **Regsvr32.exe.** (Default folder is C:\\Windows\System 32). Refer to the Windows documentation for more information on how to register a **dll**.

If both methods (“pre calculation” and “limit pre calculation”) are called, **ValueChanged** is called first. The value obtained from **ValueChanged** is then sent to **ValueLimit** (see the description of the methods below).

Properties and Methods for the Variable Pre Treatment

The interfaces for PreTreat4 listed in Table 4 have to be fulfilled by the DLL. The class methods for the interface PLCExternal5 is identical. The PLCExternal5 interface is implemented by the PreTreat3 type components.

Table 4. Interfaces

Class	Method
AdsCsRtdbPreTreat4	Initialize()
AdsCsRtdbPreTreat4	ReInitialize()
AdsCsRtdbPreTreat4	OnlineNotification()
AdsCsRtdbPreTreat4	ValueChanged(…)
AdsCsRtdbPreTreat4	ValueLimit(…)
AdsCsRtdbPreTreat4	OnControl(…)
AdsCsRtdbPreTreat4	Terminate()

Initialize

Syntax: `Initialize(obj As AdsCsInterfacesLib.IAdsCsRtdbVariable2)`

Enables the DLL to make arbitrary preparations at start-up.

Obj is a callback interface that makes it possible to read and write to signals, see also Callback Interface on page 65.

ReInitialize

Syntax: `Reinitialize()`

Enables the DLL to make arbitrary preparations after deploy. Must not run too long.

OnlineNotification

Syntax: `OnlineNotification()`

Called prior to a deploy (as opposed to *ReInitialize* that is called **after** a deploy). Lets the user application do any preparations necessary **before** a deploy.

ValueChanged

Syntax:
```
ValueChanged (sSignalName as String, vntProcessValue
As Variant, vntPreviousValue As Variant, vntNewValue As
Variant, OPCQuality As Integer, dtTimestamp as Date,
lCalculationType As Long, sPreParam As String)
```

Is called when the value of the variable is changed. **sSignalName** is the variable in the format “<controller name>:<object name>.<signal name>”.

If composite process objects are used, there may be several <object name> items in sSignalName. See also the code sample referred to in Sample Code on page 67.

vntProcessValue is the value from the Communication driver. **vntNewValue** is the value pre-treated by the DLL. **vntPreviousValue** is the previous value. **OPCQuality**: indicates the OPC quality of the variable. **dtTimestamp** is the date and time. **lCalculationType** is the selected calculation type. **sPreParam** is the entered pre calculation parameter, see Calculation Parameters for Pre Treatment on page 66. The method may assign new values to **vntNewValue** and **OPCQuality** before returning.

ValueLimit

Syntax:
```
ValueLimit(sSignalName as String, vntValue As Variant,
OPCQuality As Integer, lCalculationType As Long, sPreLimParam
As String, bEnableLimiter1 As Boolean, lTypeLimiter1 As Long,
fLimit1 As Single, fHysteresis1 As Single, bEnableLimiter2 As
Boolean, lTypeLimiter2 As Long, fLimit2 As Single,
fHysteresis2 As Single, bEnableLimiter3 As Boolean,
lTypeLimiter3 As Long, fLimit3 As Single, fHysteresis3 As
Single, bEnableLimiter4 As Boolean, lTypeLimiter4 As Long,
fLimit4 As Single, fHysteresis4 As Single, bActivateAlarm1As
Boolean, bActivateAlarm2 As Boolean, bActivateAlarm3 As
Boolean, bActivateAlarm4 As Boolean)
```

Is called when the value of the variable is changed. **sSignalName** is the variable in the format “<controller name>:<object name>.<signal name>”. .

If composite process objects are used, there may be several <object name> items in sSignalName. See also the code sample referred to in Sample Code on page 67.

vntValue is the value. **OPCQuality**: indicates the OPC quality of the variable. **lCalculationType** is the selected calculation type. **sPreLimParam** is the entered limit pre calculation parameter, see Calculation Parameters for Pre Treatment on page 66. The following 12 parameters are the default data in the database. The method is responsible for assigning values to **bActivateAlarmx** and **OPCQuality** before returning. If **bActivateAlarmx** is true, the alarm is activated after returning.

OnControl

Syntax: `OnControl(sSignalName as String, vntValue As Variant, vntPreviousValue As Variant, vntNewValue As Variant, OPCQuality As Integer, dtTimestamp as Date, lCalculationType As Long, sPreParam As String)`

Is called when the variable value is changed by a client. **sSignalName** is the variable in the format “<controller name>:<object name>.<signal name>”.

If composite process objects are used, there may be several <object name> items in sSignalName. See also the code sample referred to in Sample Code on page 67.

vntValue is the value that the client has set. **vntPreviousValue** is the previous value. **OPCQuality**: indicates the OPC quality of the variable. **dtTimestamp** is the date and time. **lCalculationType** is the selected calculation type. **sPreParam** is the entered pre calculation parameter, see Calculation Parameters for Pre Treatment on page 66. The method may assign a new value to **vntNewValue** before returning.

Terminate

Syntax: `Terminate()`

Enables the DLL to make arbitrary preparations at termination.

Callback Interface

The callback interface has two methods for accessing the RTDB: ReadValue and WriteValue.

ReadValue

Syntax`: ReadValue (sVarName As String, iProperty As Long, vntValue As Variant, OPCQuality As Integer, dtTimestamp as Date) As Long`

Read the property value of a variable. Returns less than zero if the variable cannot be read.

sVarName(in) is the variable name in the format “<controller name>:<object name>.<signal name>”.

If composite process objects are used, there may be several <object name> items in sSignalName. See also the code sample referred to in Sample Code on page 67.

iProperty(in) is the property number in the signal PCA. “1” is the value property of the signal. **vntValue**(out) returns the value of the property. **OPCQuality**: indicates the OPC quality of the variable. **dtTimestamp** is the date and time.

WriteValue

If you write data to a signal connected to an external IO then that data will be written to the controller.

Syntax`: WriteValue (sVarName As String, iProperty As Long, vntValue As Variant, OPCQuality As Integer, dtTimestamp as Date) As Long`

Writes the property value of a variable. Returns less than zero if the variable cannot be written.

sVarName(in) is the variable name in the format “<controller name>:<object name>.<signal name>”.

If composite process objects are used, there may be several <object name> items in sSignalName. See also the code sample referred to in Sample Code on page 67.

iProperty(in) is the property number in the signal PCA. “1” is the value property of the signal. **vntValue**(in) is the value of the property. **OPCQuality**(in): indicates the OPC quality of the variable. **dtTimestamp** is the date and time.

Configure Calculation Parameters

Calculation Types

Calculation types are user defined private data. It is possible to select a type when you configure calculation parameters for a signal. If desired, you can use the selected calculation types to determine the type of function (in your own application) to be performed for a given signal. By selecting a certain type of calculation, the same type of calculation (for example linearization) can be performed on a collection of signals without having to keep a record of exactly which signal to calculate.

Calculation Parameters for Pre Treatment

Calculation parameters are user defined private data. These parameters can have arbitrary contents, and can be used to parameterize a certain calculation (in your own application).

Table 5. Calculation Parameters

Parameter	Type	Comment
Pre calculation parameter	string	Sent to the variable pre treatment interface. Used in the method ValueChanged (the parameter **sPreParam**).
Limit pre calculation parameter	string	Sent to the variable pre treatment interface. Used in the method ValueLimit (the parameter **sPreLimParam**).

To configure calculation parameters:

You cannot perform calculations on signals that are SoftPoints.

1. In Plant Explorer, select the Control structure.
2. Select the relevant signal.
3. Select the Signal Configuration aspect.
4. Select the Common tab, and then configure calculation parameters. Click **Show Help** for more information.

Sample Code

The C++ project, AdsPreTreat.vcproj, contains a code sample.
The project can be found in the folder **…\ABB Industrial IT\Operate IT\PLC Connect\Samples\PreTreat4**.

PLC Connect Properties

Introduction

Most properties of the different configuration aspects can be accessed from outside PLC Connect, for example via OPC or an external program such as Bulk Data Manager. Some properties can only be viewed, others can be updated with new values.

Accessible Properties

Accessible properties of the configuration aspects are described below:

- Location: indicates where in the aspect the property is set or its value viewed when you work in Plant Explorer.
- Writable: if the property is writable or not, that is, if its value can be changed.

Alarm Event Settings Aspect

Table 6. Alarm Event Setting Parameters

Property Name	Location	Data Type	Writable
Server Data Event tab			
LimitAutoDisable	Limit auto disable box	Integer	Yes

Summary Alarm Configuration Aspect

Table 7. Summary Alarm Parameters

Property Name	Location	Data Type	Writable
GCNGuid	Generic Control Network	String	Yes
Address	Address box	String	Yes

Alarm Text Group Aspect

Table 8. Alarm Text Parameters

Property Name	Location	Data Type	Writable
Acknowledge	Acknowledge box	String	Yes
Disable	Disable box	String	Yes
Autodisable	Autodisable box	String	Yes
Enable	Enable box	String	Yes
EventOn	Event On box	String	Yes
EventOff	Event Off box	String	Yes

Operator Actions Text Groups Aspect

Table 9. Operator Actions Text Groups Parameters

Property Name	Location	Data Type	Writable
Control	Control box	String	Yes
ControlOn	Control On box	String	Yes
ControlOff	Control Off box	String	Yes
DisableControl	Disable Control box	String	Yes
EnableControl	Enable Control box	String	Yes
Force	Force box	String	Yes
ForceOn	Force On box	String	Yes
ForceOff	Force Off	String	Yes
Unforce	Unforce box	String	Yes

PLC Controller Configuration Aspect

Table 10. PLC Controller Configuration Parameters

Property Name	Location	Data Type	Writable
Protocol settings tab			
CommInfo	Communication information box	String	Yes
DriverInfo	(not shown)	String	No
FileName	(not shown)	String	Yes
AlarmOwner	Alarm Owner box	Boolean	Yes
Redundancy setting			
Failover		Integer	Yes
SlaveReadyLevel		Integer	Yes

Process Object Configuration Aspect

Table 11. Process Object Configuration Parameters

Property Name	Location	Data Type	Writable
AlarmOwner	Alarm owner box	Boolean	Yes
UserDefProcessObjectValues	User default process object values	Boolean	Yes
ObjectInfo		String	Yes

Signal Configuration Aspect

Table 12. Signal Configuration Parameters

Property Name	Location	Data Type	Writable
IsInstance (wether it is a signal or signal type)	(not shown)	Boolean	No
ID tab			
ItemType (Binary = 0, Integer = 1, Real = 2, String = 3)	Variable type box	Integer	No
ItemUsage (Internal = 0, External = 1)	Not Connected/Connected option buttons	Integer	Yes
Address	Address box	String	Yes
Common tab			
UseDefSignalValues	Use default signal values check box	Boolean	Yes
Inverted	Inverted check box	Boolean	Yes
UpdateInterval	Update interval box	Integer	Yes
Calculations dialog box			
CalcEnabled	Accessible for data access check box	Boolean	Yes
CalcTypeNo	Calculation type drop-down list	Integer	Yes
CalcEdge (Lading = 0, Trailing = 1, Both = 2)	Edge selection box	Integer	Yes
CalcHysteresis	Hysteresis box	Real	Yes
CalcParameter	Left-hand Parameter box	String	Yes
CalcEnablePreCalc	Enable pre treatment check box	Boolean	Yes

Table 12. Signal Configuration Parameters (Continued)

Property Name	Location	Data Type	Writable
CalcPreParameter	Parameter box below Enable pre treatment check box	String	Yes
CalcEnableLimCalc	Enable limit pre treatment check box	Boolean	Yes
CalcLimParameter	Parameter box below Enable limit pre treatment check box	String	Yes
Controllable tab			
UseDefControllableValues	Use default controllable values check box	Boolean	Yes
Controllable	Is controllable check box	Boolean	Yes
OperActionsTextGroupName	Operator Actions Text Group	String	Yes
LogOpActions	Log operator actions check box	Boolean	Yes
Common - Advanced dialog box			
CtrlLimEnable	Control Limits check box	Boolean	Yes
CtrlLimHigh	High box	Real	Yes
CtrlLimLow	Low box	Real	Yes
CtrlResponseEnabled	Response check box	Boolean	Yes
CtrlResponseVar	Variable box	String	Yes
CtrlResponseDelay	Delay time box	Integer	Yes
Range tab			

Table 12. Signal Configuration Parameters (Continued)

Property Name	**Location**	**Data Type**	**Writable**
UseDefRange	Use default integer values check box or Use default real values check box	Boolean	Yes
LowLimit	Low limit box	Real	Yes
HighLimit	High limit box	Real	Yes
LowLimitPLC	Low limit in PLC box	Real	Yes
HighLimitPLC	High limit in PLC box	Real	Yes
Unit	Engineering unit box	String	Yes
Precision	Single/Double Precision option buttons	Integer	Yes

Alarm Event Configuration Aspect

The property names have an *X* at the end. For integer and real signals, *X* is a number between 1–8. For binary signal, *X* equals 1.

The property SummaryAlarm has Y at the end, where Y is a number between 1-10.

Table 13. Alarm Event Configuration Parameters

Property Name	**Location**	**Data Type**	**Writable**
Limiter tab			
UseDefLimValues*X*	Use default limiter values check box	Boolean	Yes
LimEnabledX	Use Limiter	Boolean	Yes
LimName*X*	Name box	String	Yes

Table 13. Alarm Event Configuration Parameters (Continued)

Property Name	**Location**	**Data Type**	**Writable**
LimType*X* (Max = 1, Min = 0)	Type drop-down list	Integer	Yes
Limit*X*	Limit box	Real	Yes
LimHysteresis*X*	Hysteresis box	Real	Yes
Event tab			
UseDefEventValues*X*	Use default event values check box	Boolean	Yes
IsEvent*X*	Is an event check box	Boolean	Yes
TimeStampFromPLC*X*	Time stamped from controller check box	Boolean	Yes
LogStatusChgOff*X*	Log status changes 'Off' check box	Boolean	Yes
LogStatusChgOn*X*	Log status changes 'On' check box	Boolean	Yes
AlarmTextGroupNameX	Alarm text group drop-down list	String	Yes
OverrideDefCondNameX	Override Default Condition Name check box	Boolean	Yes
CondNameX	Box under Override Default Condition Name check box	String	Yes
Event2 tab			
MMSNotifyVariable*X*	Box under the When detected notify variable check box	String	Yes
MessageX	Message text box	String	Yes
EventTextBulk*X*	Extended event text box	String	Yes

Table 13. Alarm Event Configuration Parameters (Continued)

Property Name	Location	Data Type	Writable
Alarm tab			
UseDefAlarmValues*X*	Use default alarm values check box	Boolean	Yes
IsAlarm*X*	Is an alarm check box	Boolean	Yes
AckType*X* (Normal = 0, None = 1, Disappears = 2)	Acknowledgement selection box	Integer	Yes
Severity*X*(1)	Severity box	Integer	Yes
FilterTime*X*	FilterTime box	Integer	Yes
SummaryAlarmXY	Summary alarms for this event list view	String	Yes
Alarm2 tab			
MMSAckVariable*X*	Box under the When acknowledged notify check box	String	Yes
Class*X*	Class box	Integer	Yes

(1) If you update many properties of this type at the same time, and alarm priority changes are set to be automatically deployed, then the updating can take a long time.

PLC Dial Configuration Aspect

Table 14. PLC Configuration Parameters

Property Name	Location	Data Type	Writable
IndexCorrection	Index Correction box	Integer	Yes
IndexRegister	Index Register box	String	Yes
NoOfRegisters	Number Of Registers box	Integer	Yes

Table 14. PLC Configuration Parameters (Continued)

Property Name	Location	Data Type	Writable
PhoneNumber	Phone Number box	String	Yes
PLCNo	Controller Identity box	String	Yes
Protocol	Protocol drop-down list	String	Yes
SchedTries	# tries box	Integer	Yes

Dial Log Configuration Aspect

Table 15. DialLog Configuration Parameters

Property Name	Location	Data Type	Writable
UseDefaultPLCValues	Use Default PLC Values check box	Boolean	Yes
IndexCorrection	Index Correction box	Integer	Yes
IndexRegister	Index Register box	String	Yes
NoOfRegisters	# Registers box	Integer	Yes
StartRegister	Start Register box	String	Yes
HistoryLog	History Log drop-down list	String	Yes

Example - Change Signal Properties with Bulk Data Manager

1. Start Plant Explorer.
2. Start Excel.

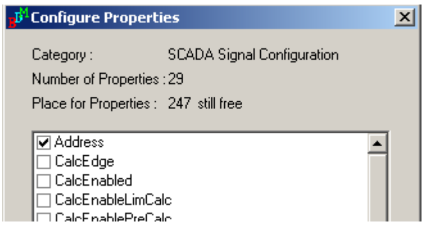
Set up Options

3. In Excel, click the **Activate BDM** button in the Bulk Data Manager toolbar.
4. Select **Bulk Data Manager > Options**.

5. Under the General tab, select to Identify Objects by ARD with path.
6. Under the Transaction tab, select By Object as Transaction mode.

Set up Properties

7. In Plant Explorer, select the Control Structure.
8. Select the desired Generic Control Network object.
9. Select the Control Structure aspect, and then drag it from Plant Explorer to cell A1 in Excel.
10. Return to Plant Explorer and select a Signal Configuration aspect. It does not matter which one.
11. Drag the aspect from Plant Explorer to the first free cell on row 1 in Excel. In this example, it is E1.
12. In the displayed dialog box, select the properties you want to configure. For example, you can select Address, Controllable, ItemUsage and UseDefControllableValues.



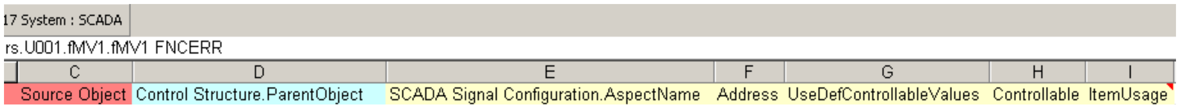
Category : SCADA Signal Configuration
Number of Properties : 29
Place for Properties : 247 still free

- [✓] Address
- [ ] CalcEdge
- [ ] CalcEnabled
- [ ] CalcEnableLimCalc
- [ ] CalcEnablePreCalc

Figure 40. Configuration Properties

13. Click **OK**.
14. In Excel, move the column UseDefControllableValues to a column in front of the Controllable column, see Figure 41. This is necessary since you have to set

the property UseDefControllableValues to False, before you can change the value of the Controllable property.



17 System : SCADA

rs.U001.fMV1.fMV1 FNCERR

C	D	E	F	G	H	I
Source Object	Control Structure.ParentObject	SCADA Signal Configuration.AspectName	Address	UseDefControllableValues	Controllable	ItemUsage

Figure 41. Column Order

Export Objects

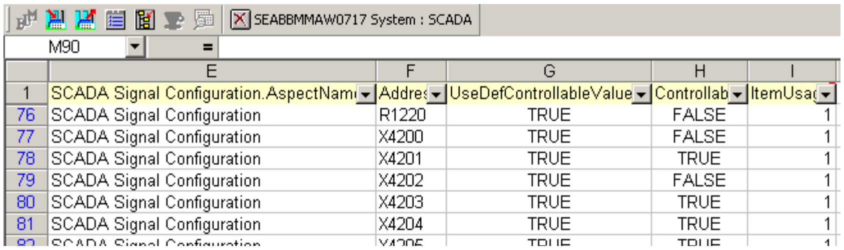
15. In Plant Explorer, select the Control Structure.
16. Select the same Generic Control Network object as in Step 8.
17. Drag the Generic Control Network object from Plant Explorer to cell A2 in Excel. The object and all objects under it will be exported.

Filter Objects

18. In Excel, select all cells.
19. Select **Data > Filter > AutoFilter**. This makes it possible to filter the objects in Excel. For example, you can filter the objects on [Control Structure.Parent Object]. Then when you select one process object, you will only see its signals.

Change Property Values

20. You can now change the property values, for example:
    - Change some Modbus addresses to Comli addresses
    - Set some signals to be controllable
    - Set ItemUsage to internal for signals with Modbus addresses. (A property value of ‘0’ means Internal, and a value of ‘1’ means External).
21. When ready, click the **Save All Objects** button in the Bulk Data Manager toolbar. The rows that the filter shows will be updated in the system.



SEABBMMAW0717 System : SCADA

M90 =

	E	F	G	H	I
1	SCADA Signal Configuration.AspectNam	Addres	UseDefControllableValue	Controllab	ItemUsa
76	SCADA Signal Configuration	R1220	TRUE	FALSE	1
77	SCADA Signal Configuration	X4200	TRUE	FALSE	1
78	SCADA Signal Configuration	X4201	TRUE	TRUE	1
79	SCADA Signal Configuration	X4202	TRUE	FALSE	1
80	SCADA Signal Configuration	X4203	TRUE	TRUE	1
81	SCADA Signal Configuration	X4204	TRUE	TRUE	1
82	SCADA Signal Configuration	X4205	TRUE	TRUE	1

Figure 42. Property Values

Dial Manager Server Access

License Information: The dial manager is part of the PLC Connect Dialed communication functionallity which requires a separate license to run.

Introduction

The Dial Manager Server is handling dialed communication, including initiating and terminating calls requested from the Dial Panel or Dial Supervisor.

How to configure dialed communication is described in the Section 5, Configure Dialed Communication.

Via the Dial Manager interface it is possible for a user written standalone application to initiate and disconnect new calls. The application can be with or without a user interface, and written in, for example, Visual Basic or C++. It can be run on both server and client nodes.

Prioritized Calls

Prioritized calls are only accessible via the Dial Manager interface, and not at all via the PLC Connect standard aspects Dial Panel and Dial Supervisor. Hence, in order to make these call types available you must do this via a user written application. When writing such an application you should take great care, and be aware of the fact that calls in progress are **disconnected unconditionally**. Logical errors in such an application can for example break an ongoing collection of historic data from PLCs, or cause other unwanted behavior.

Prioritized calls makes it possible to interrupt other calls in progress, for example when there are no free connections (lines). When interrupted, a call will **not** be automatically re-established. Calls in progress will be disconnected as follows:

1. Time Limit
2. Manual Close
3. Permanent
4. History: Single Controller
5. Cyclic
6. Prioritized

Accessible Methods

The Dial Manager interface belongs to the library ADSDIALMANAGERHDLRLib, and the accessible methods to the class AdsDialMgrHandler.

ConnectPLC

Use this method to initiate a call to a controller.

Syntax: `ConnectPlc(sPLCName As String, TypeOfCall As DMTypeOfCall, lRunTimeMinutes As Long, bKeepTextFiles As Boolean)`

sPLCName (in): controller name.
TypeOfCall (in): type of call, see also Table 16.
lRunTimeMinutes (in): the time limit in minutes for time limited calls. For other

call types this parameter should be zero (0).
bKeepTextFiles (in): whether or not to keep text files after an import (for debugging purposes only). This parameter is only relevant for calls of type “History: Single Controller”. For other call types this parameter should be False.

Table 16. Accessible Call Types

Type of call	Enumeration
Manual Close	ManualNormalDMTypeOfCall
Time Limit	ManualTMODMTypeOfCall
Permanent	ManualPermanentDMTypeOfCall
History: Single Controller	ManualHistoryDMTypeOfCall
Prioritized	PrioritizedDMTypeOfCall

DisconnectPLC

Use this method to disconnect a call.

Syntax: `DisconnectPLC(lCTNReq As Long) As Boolean`

Returns FALSE if the call could not be disconnected.

lCTNReq (in): CTN number (connection or line identifier) for the call.

GetLineStatus

Use this method to get information on the status of relevant connections (lines).

Syntax: `GetLineStatus(lNoOfLines As Long, lNoOfCyclInQueue As Long, sLineName() As String, lTimeConnected() As Long, TypeOfCall() As DMTypeOfCall, sPLCName() As String, sOperator() As String, CallStatus() As DMCallInProgres, CTN() As Long, bAbortPoss() As Boolean, bEnablePoss() As Boolean, bDisablePoss() As Boolean, bAnyCyclicCalls As Boolean)`

lNoOfLines (out): number of lines.
lNoOfCyclInQueue (out): number of queued cyclic calls.

sLineName (out)(array): line names.
lTimeConnected (out)(array): how long time each call have been established.
TypeOfCall (out)(array): call type for each line.
sPLCName (out)(array): controller name for each line.
sOperator (out)(array): For future use.
CallStatus (out)(array): status for each call.
CTN (out)(array): CTN numbers, that is, call identifiers.
bAbortPoss (out)(array): whether or not calls can be aborted.
bEnablePoss (out)(array): whether or not calls can be enabled.
bDisablePoss (out)(array): whether or not calls can be disabled.
bAnyCyclicCalls (out): whether or not there are cyclic calls in progress.

Notes on PLC Connect Real Time Data Access

The interface to the Dial Manager server can be used together with the interface described in PLC Connect Real Time Data Access on page 42, for example to write data to the RTDB.

Typically, it can work as follows:

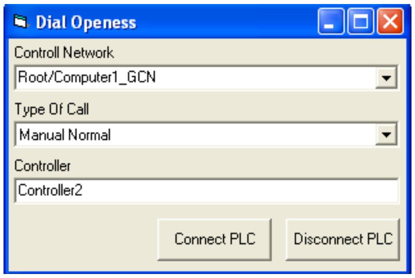
1. Use subscription to wait for real time events. Subscription is available in the Real Time Data Access interface.
2. When the expected event occurs, check the status of relevant connections (lines) with the GetLineStatus method.
3. If the connection already is established, then you can perform the desired operations, for example write, straight away using the Real Time Data Access interface.

   If the connection isn't established, first use the ConnectPLC method to initiate a call, then perform the desired operations.
4. Disconnect from the controller with the DisconnectPLC method.

Sample

This is a Visual Basic 6.0 sample, **DialManagerOpenness.vbp**, which can be found in the directory **\...\ABB Industrial IT\Operate IT\PLC**

Connect\Samples\DialManagerOpenness. Figure 43 shows the sample user interface.



Controll Network: Root/Computer1_GCN

Type Of Call: Manual Normal

Controller: Controller2

Figure 43. Dialog Box Used in Sample

Section 5 Configure Dialed Communication

License Information: The dial manager is part of the PLC Connect Dialed communication functionallity which requires a separate license to run.

Introduction

This sub-section describes how to configure dialed communication:

- Basic Preparations on page 86 describes what you need to do before you can start the configuration.
- Basic Configuration Example on page 88 describes an example configuration, not including dialed history.
- Dialed History on page 100 gives an overview of how dialed history works in relation to dialed controllers.
- Dialed History Configuration Example on page 103 describes an example configuration of dialed history.

What is Dialed Communication?

Remote controllers can be dialed up and communicated with via Comli or Modbus (Serial), that is, alarm, event and historic information can be collected to a server. Provided are:

- Manually order calls, with and without a time limit.
- Permanent calls.
- Scheduled (cyclic) calls for collection of historic data.

Incoming connections are also supported, where controllers can call up the server in the case of an alarm and so on.

Basic Preparations

Before you can configure or use dialed communication, you have to start the Dial Manager server, set up the modems to use, and create the relevant objects and signals.

Add Dial Manager Server Process

1. Add the Dial Manager server process to the PLC Server. For more information on how to do this, refer to the information on PLC Server in the PLC Connect online help.

Setup Modems

2. To be able to dial up remote controllers, each communication port you want to use has to be connected to a modem.

For more information on how to install a modem and set up dialing rules, refer to the modem and Windows documentation.

Create Dialed Controllers

3. In Plant Explorer, select the desired Generic Control Network object.
4. Create the relevant dialed controllers for the control network. Refer to the PLC Connect online help for more information on how to create a dialed controller.
5. Repeat Step 3 to Step 4 for each additional control network for which you want to configure dialed communication.

Set up Communication Monitoring

6. Import the process object type CommunicationType. It can be found in the file **\...\ABB Industrial IT\Operate IT\PLC Connect\Data\ObjectTypes\CommunicationType.afw**. For more

information on export and import, refer to the *Industrial IT 800xA - System Administration and Security (3BSE037410*).*

7. The signal Bad is part of the process object type CommunicationType. Set the signal to be only an event, not an alarm. Otherwise, it may generate alarms when a call to a controller is initiated. Refer to the PLC Connect online help for more information on how to set a signal to be an event and/or alarm.

Process objects of this type are used to monitor the communication status for a dialed controller. Generally, it is hard to get informed when a cyclic call fails. However, the signal Error in process objects of type CommunicationType will generate an alarm when a cyclic call fails.

8. For one of the dialed controllers, create a process object based on the process object type CommunicationType. Refer to the PLC Connect online help for more information on how to create process objects. Make sure to:
   – place the process object under the dialed controller you want to monitor.
   – define the object as communication, that is, select Communication when entering object information for the object.
   – set the update time to about 10 000 ms when entering object information for the object.
9. Repeat Step 8 for each additional dialed controller.

Create Process Objects and Signals

10. For each dialed controller, create the relevant process objects and signals. Refer to the PLC Connect online help for more information on how to create process objects and signals.
11. Deploy the configuration for each affected control network. To make a deploy, select the Deploy aspect of a Generic Control Network object, and then click **Deploy**.

You have completed basic preparations that are necessary for the configuration.

Basic Configuration Example

This example describes how to make a basic configuration of dialed communication for a specific control network. This configuration does not include dialed history. However, it's assumed that you already have made the necessary preparations, see Basic Preparations on page 86.

Set up Protocols

The first part is to set up the protocols to use.

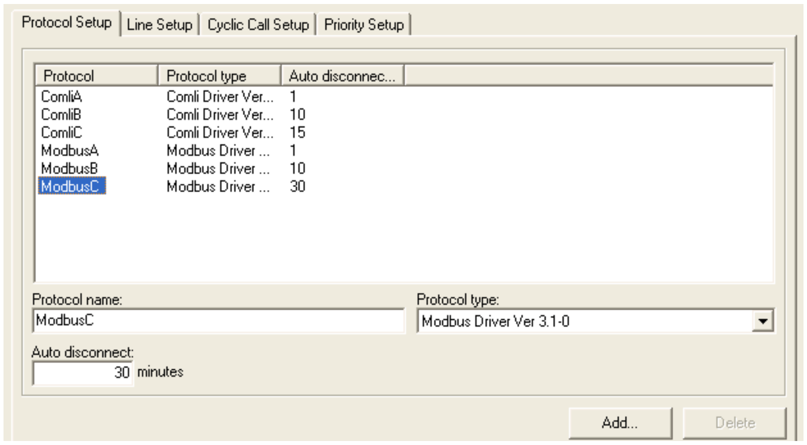
Startup

The Dial Editor aspect is used to configure dialed communication for a specific control network.

1. In Plant Explorer, select the Control Structure.
2. Select the desired Generic Control Network object.
3. Add a Dial Editor aspect.
4. Select the added aspect.
5. Select the Protocol Setup tab.

Add Protocols

Under the Protocol Setup tab, see Figure 44 for an example, there is a table with information on the defined protocols.



Protocol	Protocol type	Auto disconnec...
ComliA	Comli Driver Ver...	1
ComliB	Comli Driver Ver...	10
ComliC	Comli Driver Ver...	15
ModbusA	Modbus Driver ...	1
ModbusB	Modbus Driver ...	10
ModbusC	Modbus Driver ...	30

Protocol name: ModbusC
Protocol type: Modbus Driver Ver 3.1-0
Auto disconnect: 30 minutes

Figure 44. Protocol Setup Tab

6. Under the tab, click **Add**. The New Protocol dialog box is displayed.
7. Enter a protocol name.
8. Click **OK**. The dialog box is closed, and the protocol added to the table.

Configure the Protocol

9. Select the added protocol.
10. From the Protocol type drop-down list, select the type of protocol.
11. In the Auto disconnect box, enter the time in minutes after which a manually initiated call using this protocol is to be terminated by default.
12. Click **Apply**.

Configure the COM-port

13. Configure the following for each COM-port that uses the protocol:

    Use the Control Panel in Windows to enter these settings. Refer to the Windows documentation for more information.

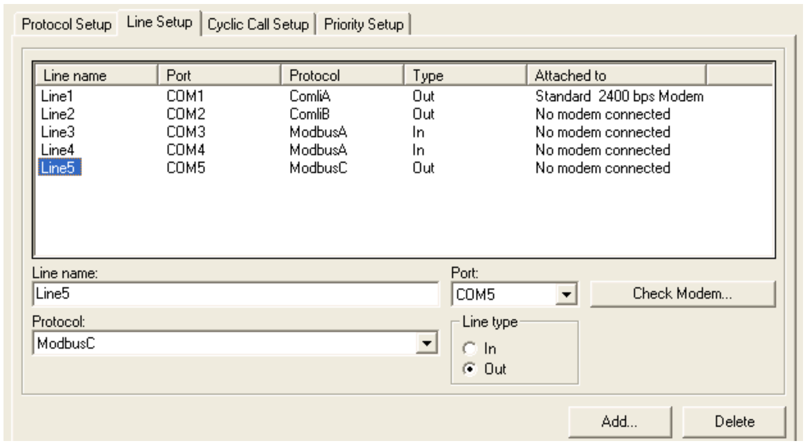
    – Communication speed (Baud-rate).
    – Number of data bits. For Comli and Modbus protocols, this number is fixed to 8.
    – Number of stop bits. For Comli and Modbus protocols, this number is fixed to 1.
    – Parity setting.
    – Flow control setting. For Comli and Modbus protocols, the flow control is fixed at “None”.

14. If desired, repeat Step 6 to Step 13 for each other protocol you want to add.
15. Select the Line Setup tab.

Set up Connections

The next part is to set up the connections (lines) to use.

Under the Line Setup tab, see Figure 45 for an example, there is a table with information on the defined connections.



Line name	Port	Protocol	Type	Attached to
Line1	COM1	ComliA	Out	Standard 2400 bps Modem
Line2	COM2	ComliB	Out	No modem connected
Line3	COM3	ModbusA	In	No modem connected
Line4	COM4	ModbusA	In	No modem connected
Line5	COM5	ModbusC	Out	No modem connected

Line name: Line5
Port: COM5
Protocol: ModbusC
Line type: In [✓ Out]

Figure 45. Line Setup Tab

The Attached to column shows the type/name of the modem connected to the selected communication port, that is, the modem name as defined in the Windows Control Panel.

Add a Connection

1. Under the tab, click **Add**.
2. Enter a name in displayed dialog box.
3. Click **OK**. The dialog box is closed, and the connection added to the table.

Configure the Connection

4. Select the added connection in the table.
5. Select a COM port from the Port drop-down list.

6. Select a defined protocol from the Protocol drop-down list.
7. In the Line type area, select either:
   - **In** to dedicate the connection to incoming calls.
   - **Out** to dedicate the connection to outgoing calls.
8. Click **Apply**.

Check the Modem

The **Check Modem** button is used to check if a modem is connected and switched on.

9. Select the configured connection in the table.
10. Click **Check Modem**. A dialog box will show the result, see Table 17 for examples.

Table 17. Modem Check Examples

Answer	Description
"Modem answered OK"	The text string "OK" was received as a reply.
"No answer from modem"	Either the modem is not connected or the power is off.
"Modem answers incorrectly"	Data was received, but could not be interpreted.
"Invalid port number"	The selected port number does probably not exist in the computer.

11. If desired, repeat Step 1 to Step 10 for each other connection you want to add.
12. Select the Cyclic Call Setup tab.

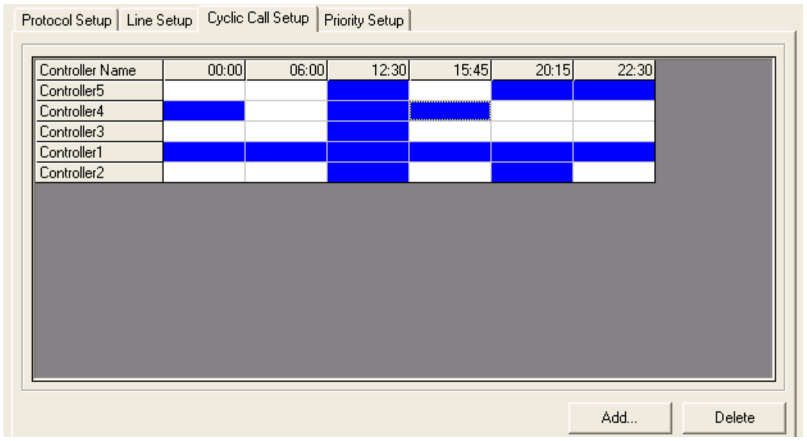
Set up Cyclic Calls

The next part is to set up the cyclic calls to use.

Cyclic calls are used to automatically collect data, including historic data, from a number of controllers at scheduled times. For example, at 3 PM the data can be scheduled to be collected from controllers A and B, while at 5 PM it can be scheduled to be collected from controllers C and D. These scheduled collections will be repeated every day until changed.

Under the Cyclic Call Setup tab, see Figure 46 for an example, there is a table with information on the defined cyclic calls. The table contains:

- a column with the dialed controllers.
- a column for each defined call time (maximum is 24).



Controller Name	00:00	06:00	12:30	15:45	20:15	22:30
Controller5			🟦		🟦	🟦
Controller4	🟦		🟦	🟦		
Controller3			🟦			
Controller1	🟦	🟦	🟦	🟦	🟦	🟦
Controller2			🟦		🟦	

Figure 46. Cyclic Call Setup Tab

An active (blue) cell in the table means that the controller is scheduled to be called at the call time shown in the column header.

Add a Call Time

1. Under the tab, click **Add**.
2. Enter the call time in hours and minutes in the displayed dialog box.
3. Click **Apply** in the dialog box. A new call time column is added to the table.
4. If desired, repeat Step 2 to Step 3 for each other call time you want to add.
5. When ready, click **OK**. The dialog box is closed.

Schedule Calls

6. Click the cell in the table corresponding to the desired call time for a controller. The cell becomes active (blue).

When you want to clear a scheduled call, click in the cell again.

7. Click **Apply**.
8. Right-click a cell in the table.
9. From the displayed context menu, select Set Column. All cells in the column will be scheduled as calls.

You can use the context menu to edit call times, schedule or clear multiple calls, and so on.

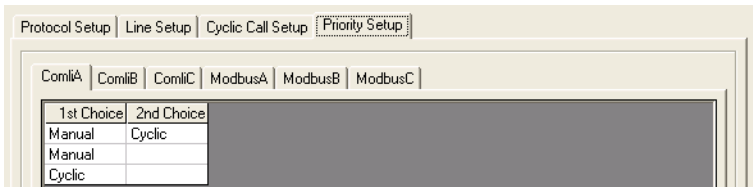
10. Click **Apply**.
11. If desired, schedule more calls as described above.
12. Select the Priority Setup tab.

Set up Priorities

The next part is to set up priorities.

Priorities are used to avoid dedicating modems for either manual or cyclic calls. This way it is possible to allocate many modems for cyclic calls when no manual calls are in progress and vice versa. Modem failures can also be handled by defining how the remaining modems are to be used.

Under the Priority Setup tab, see Figure 47 for an example, there is a tab for each defined protocol with information on the priorities of the outgoing connections using that protocol.



1st Choice	2nd Choice
Manual	Cyclic
Manual	
Cyclic	

Figure 47. Priority Setup Tab

The number of priority levels corresponds to the number of modems handling cyclic and manual calls, that is, the number of outgoing connections using the protocol. For each priority level you define the primary use (1st choice) as Cyclic or Manual and a secondary use (2nd choice) as Cyclic, Manual or "Blank".

The settings are for priority levels and **not** for specific modems.

The top-most row corresponds to the top-most priority, and so on downwards.

Set or Change Priority Levels for a Protocol

1. Under the Priority Setup tab, select the tab for the protocol.
2. In the first row, click in the 1st choice column to change the primary use.
3. In the first row, click in the 2nd choice column to change the secondary use.
4. Repeat Step 2 to Step 3 for each additional row.
5. Click **Apply**.
6. Repeat Step 1 to Step 5 for each other protocol tab.
7. Review the priority guidelines in the next sub-section, or skip directly to the next part, see Set up Dialed Controllers on page 98.

Priority Guidelines

- Begin by setting up the top-most row with Manual as 1st choice and Cyclic as 2nd.
- If more than one outgoing line is available consider to set up a row with Manual as 1st choice and an empty 2nd choice, to make sure one connection is reserved for manual calls.
- If several lines are available and it's important that cyclic calls are not blocked by manual calls a row with Cyclic as 1st choice and 2nd choice empty can be set up.
- If a lot of lines are available the remaining can be defined as Manual as 1st choice and Cyclic as 2nd choice.

Basic Example

In this example, see Table 18, five connections have been defined as outgoing for a protocol:

- One modem has been reserved for manual calls only (no 2nd choice).
- One modem has been reserved for cyclic calls only (no 2nd choice).
- When needed, the modems on the top-most and bottom-most rows are used for manual calls, when no manual calls are requested, they can be used for cyclic calls.
- When needed, the modem on the second top-most row is used for cyclic calls, when no cyclic calls are requested, it can be used for manual calls.

Table 18. Five Outgoing Connections

1st choice	2nd choice
Manual	Cyclic
Cyclic	Manual
Manual	
Cyclic	
Manual	Cyclic

Assume that four cyclic calls are in progress and four manual requests are issued, which requires the use of more modems than are currently available.

1. The first manual call is served immediately, as one modem has been reserved for manual calls only.
2. The second manual request is queued and will get a modem as soon as one of the cyclic calls completes.
3. The other two manual requests are not queued and the operator is informed that the manual requests cannot be served, because cyclic calls are in progress and no more modems are available.

Example - One Outgoing Connection

In this example, see Table 19, one connection has been defined as outgoing for a protocol. Cyclic calls can be in progress when no manual calls are in progress.

Table 19. One Outgoing Connection

1st choice	2nd choice
Manual	Cyclic

Example - Three Outgoing Connections

In this example, see Table 20, three connections has been defined as outgoing for a protocol:

Table 20. Three Outgoing Connections

1st choice	2nd choice
Manual	Cyclic
Cyclic	Manual
Manual	

- When three manual calls are in progress, the calling of cyclic scheduled controllers start when any of the manually started calls in progress are stopped. When the first cyclic scheduled controller has been called, it is not possible to have more than two manual calls in progress before the queue of cyclic scheduled controllers is empty.
- When max one manual call is in progress, two cyclic scheduled controllers can be called simultaneously.
- One line is always reserved for manual calls.

Example - Many Outgoing Connections

In this example, see Table 21, many connections has been defined as outgoing for a protocol:

Table 21. Many Outgoing Connections

1st choice	2nd choice
Manual	Cyclic
Cyclic	Manual
Manual	
Manual	Cyclic
Cyclic	
(...)	(...)
(Manual)	(Cyclic)

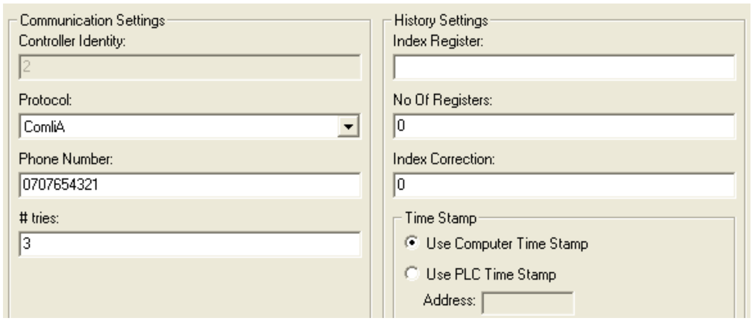
- One line is always reserved for manual calls (no 2:nd choice), and one for cyclic calls (no 2:nd choice).
- When max one manual call is in progress, four lines can simultaneously be used for cyclic calls if needed.
- If four manual calls are in progress when several controllers are to be called cyclic, one line is always available for cyclic calls. After the first manual call has stopped, the controllers queued for cyclic calls keep two lines until the queue is empty (the rows with cyclic as 1:st choice). Max three manual calls can be in progress as long as the queue of cyclic called controllers isn't empty.
- When more lines are available they can be added with Manual as 1st choice and Cyclic as 2nd.
- If it is not so important that the calling of cyclic scheduled controllers starts immediately the row with Cyclic as 1:st and no 2:nd choice can have Manual as 2nd choice. The difference is that the calling of cyclic controllers starts on one line when the first manual call stops.

Set up Dialed Controllers

The next part is to set up the dialed controllers in the control network.

1. In the Control Structure, select a dialed controller part of the Generic Control Network object.

2. Select the PLC Dial Configuration aspect, see Figure 48 for an example.


Communication Settings
Controller Identity: 2
Protocol: ComliA
Phone Number: 0707654321
# tries: 3

History Settings
Index Register:
No Of Registers: 0
Index Correction: 0
Time Stamp
(●) Use Computer Time Stamp
( ) Use PLC Time Stamp
Address:

Figure 48. PLC Dial Configuration Aspect

In the aspect, you can enter communication settings for the dialed controller and also default history settings. These history settings will be set later on as part of the dialed history configuration example.

Enter Communication Settings

3. From the Protocol drop-down list, select a defined protocol to use.
4. In the Phone Number box, enter the phone number to be dialed to contact the controller. The number should be entered without spaces, for example, “0707 654321” should be entered as “0707654321”.

Some switchboards require a Wait command before you get an outside line, for example, 0W01234567.

5. In the #tries box, enter the number of dial attempts for a cyclic call sequence that is to be carried out before the call is considered to have "failed".
6. Click **Apply**.
7. Repeat Step 1 to Step 6 for each other dialed controller in the control network.

Finish the Basic Configuration Example

8. Deploy the configuration for the control network. For more information on how to deploy a configuration, refer to the PLC Connect online help.
9. Repeat the previous steps in the basic configuration example for each other control network you want to set up dialed communication for.
10. The configuration is now finished for both incoming and manually ordered calls. To also configure for dialed history, continue with Dialed History Configuration Example on page 103.

Dialed History

Dialed history is used to collect historic data from process signals that don’t require a permanent connection, for example, for signals used for statistic analysis or status surveillance.

The server calls a controller at scheduled times to collect the historic data. These calls are denoted cyclic calls. A collection can also be ordered manually with the Dial Supervisor.

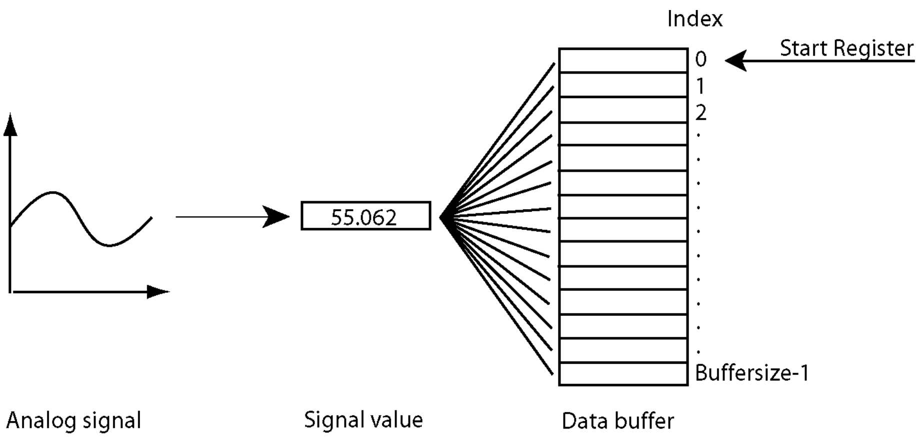


Figure 49. Signal Sampling

The dialed controller has to be programmed to sample each signal value periodically and store the value in a circular data buffer, see Figure 49. When a value is stored in a circular data buffer, the oldest sample value is overwritten with the new sample value. There should be one data buffer for each signal. The required buffer size depends on how often the values are collected. For example, a signal is sampled and stored every half hour and the values are collected once every day. The data buffer then should be configured to contain 48 values.

Create one index register in the controller for each data buffer to keep track of the latest sampled value in a buffer. The index register value should be incremented each time a new value is stored in the data buffer.

Depending on how the controller is programmed, the index register can hold a number between 0 and [buffer size - 1], or between 1 and [buffer size]. In the first case, the index correction has to be 0; in the second case, it has to be 1.

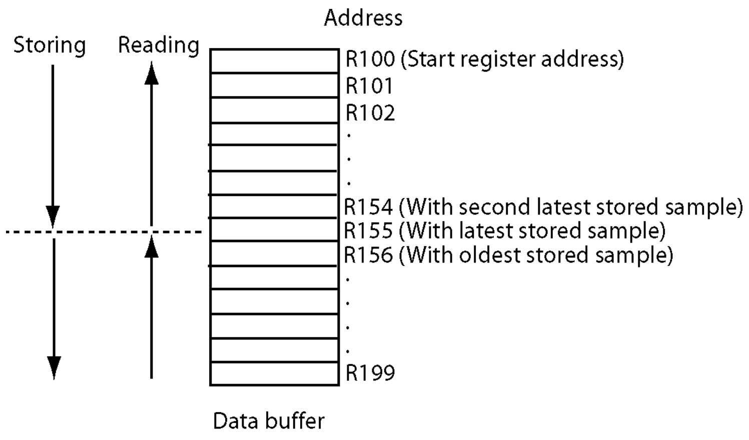

Address
Storing
Reading
R100 (Start register address)
R101
R102
R154 (With second latest stored sample)
R155 (With latest stored sample)
R156 (With oldest stored sample)
R199
Data buffer

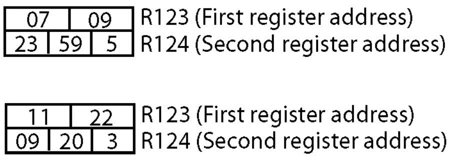
Figure 50. Controller Data Buffer

The index register value together with the index correction value and the buffer start register address are used to find the stored sampled values in the controller, see Figure 50. If for example the index register holds the value 55, then this is the latest stored value and the reading starts on the following. That is, if the start address is 100, then the reading is to start on R155, go backwards through R154 to the start address and then continue from the start address + number of registers backwards to R156. This is if the Index correction is 0. When the Index correction is 1, reading starts on 154. An index correction of 0 means that the index register holds a value between 0 and the number of registers - 1. When the index correction is 1 then the index register holds a value between 1 and the number of registers.

The maximal number of values that can be read each time is 32. If the number of registers is more than that they are read 32 registers at a time until all have been read.

PLC Time Stamp

When historic data is collected, the time of the last sample can be time stamped in two ways. By default, the time for the call to the controller is used as time of the last sample. When PLC time stamp is used, instead the time of the last sample is read from two registers in the controller.



07 09
23 59 5
R123 (First register address)
R124 (Second register address)
11 22
09 20 3
R123 (First register address)
R124 (Second register address)

Figure 51. PLC Time Stamp Registers

The two registers in the controller should be of data type integer (16 bits), and the first register should hold the date in the format MMDD. MM is the month and DD is the day. For example, 0709 equals 9 July and 1122 equals 22 November. If the date in the controller is later than the date in the computer, then the previous year is used.

The second register should hold the time in the format HHMMS. HH is the hour, MM is the minute, and S is tens of seconds. For example, 23595 equals 23:59:50, and 09203 equals 09:20:30. Midnight is 00000.

The time stamp from controller is only available when the selected controller type and communication protocol permit this. The valid communication protocols are Comli (telegram type “Å” and “Ä”) and Sattbus. It is not displayed if OPC is used.

Dialed History Configuration Example

This example describes how to make a configuration of dialed history. However, it is assumed that the basic configuration example already is performed, see also Basic Configuration Example on page 88.

Preparations

The first part contains some preparations.

Import DialHistoryType

1. Start the Import/Export tool. Refer to the *Industrial IT 800xA - System Administration and Security (3BSE037410*)* for more information on the Import/Export tool.
2. Use the Import/Export tool to import the process object type DialHistoryType. It can be found in the file **\...\ABB Industrial IT\Operate IT\PLC Connect\Data\ObjectTypes\DialHistoryType.afw**.

Create DialHistoryType Objects

3. In Plant Explorer, create a process object of the process object type DialHistoryType for one of the dialed controllers. Refer to the PLC Connect online help for more information on how to create process objects. Make sure to:

   – name the process object with a capital letter “X” followed by the controller name. For example, if the controller name is “Dial_TH1”, then name the object “XDial_TH1”.

When the controller name is longer than 22 characters, use only the first 22 characters.

   – define the object as a structure, that is, select Structure when entering object information for the object.

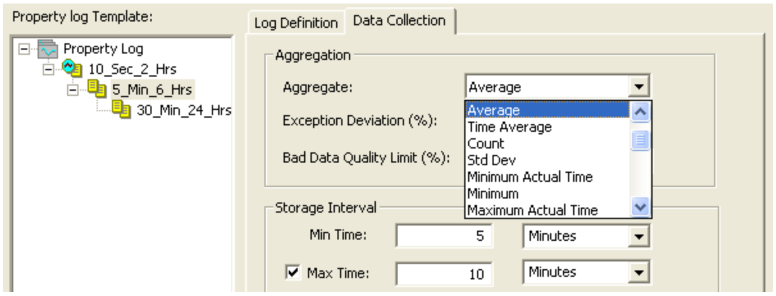
The signals in the created object are not connected. **Don’t** change their initial configuration.

4. Repeat Step 3 for each additional dialed controller part of any Generic Control Network object.

Configure History Logging

5. Configure history logging for each signal part of a dialed controller. Refer to the Industrial IT 800xA - System Operator Workplace Configuration (3BSE030322Rxxxx) for more information on history logging.

If you wish to use hierarchical logs based on a dialed history direct log (see Figure 54 for an example), make sure to select either **Average, Maximum or Minimum** as aggregation (see Figure 52 for an example). For dialed history, only these aggregations are supported.



Aggregate: Average
Exception Deviation (%):
Bad Data Quality Limit (%):
Storage Interval
Min Time: 5 Minutes
[✓] Max Time: 10 Minutes

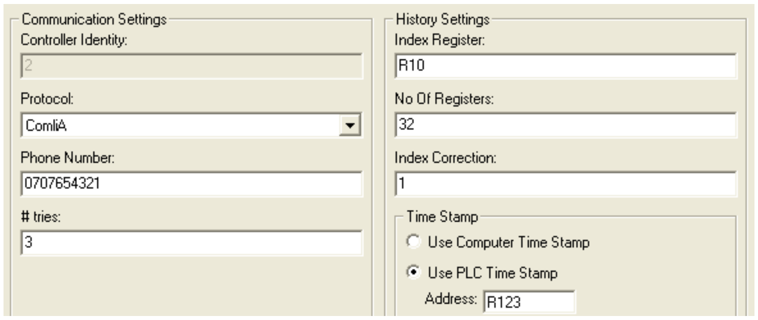
Figure 52. Hierarchical Log Configuration

6. Deploy the configuration for each affected control network.

Enter Default History Settings

The next part is to enter default history settings for a dialed controller.

1. In the Control Structure, select a dialed controller.
2. Select the PLC Dial Configuration aspect, see Figure 53 for an example.



Controller Identity: 2

Protocol: ComliA

Phone Number: 0707654321

tries: 3

Index Register: R10

No Of Registers: 32

Index Correction: 1

( ) Use Computer Time Stamp

(●) Use PLC Time Stamp

Address: R123

Figure 53. Entered History Settings

In the aspect, you can enter default history settings. The communication settings were entered earlier on as part of the basic configuration example.

The index register address, data buffer size, index correction and PLC time stamp register address mentioned below depend on how the dialed controller is programmed. For more information, see Dialed History on page 100.

3. In the Index Register box, enter the index register address. The index register is the register used in the dialed controller to keep track of where the latest historic value for the signal is stored.
4. In the No Of Registers box, enter the number of registers used in the dialed controller to store the historic values for the signal. This number is also called the data buffer size.
5. In the Index Correction box, enter the index correction value. This value is used to offset where the latest historic value is stored. It is usually equal to 0 or 1.
6. Select either:
   - Use Computer Time Stamp. When historic data is collected, the time for the call to the controller is used as time of the last sample.
   - Use PLC Time Stamp. When historic data is collected, the time of the last sample is read from two registers in the controller. Enter the address for the first of these two registers in the Address box.

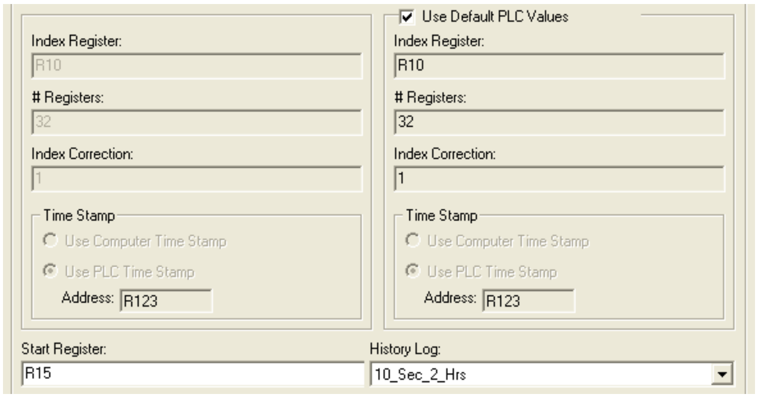
7. Click **Apply**.
8. Repeat Step 1 to Step 6 for each other dialed controller in the control network.

Enter Signal History Settings

The last part is to enter history settings for a signal part of a dialed controller.

1. In the Control Structure, select the signal part of a dialed controller.
2. Add a Dial Log Configuration aspect.
3. Select the added aspect.

In the aspect, you can enter signal history settings, see Figure 54 for an example.


[✓] Use Default PLC Values

Index Register: R10

Registers: 32

Index Correction: 1

Time Stamp

Use Computer Time Stamp [✓ Use PLC Time Stamp]

Address: R123

Index Register: R10

Registers: 32

Index Correction: 1

Time Stamp

Use Computer Time Stamp [✓ Use PLC Time Stamp]

Address: R123

Start Register: R15

History Log: 10_Sec_2_Hrs

Figure 54. Dial Log Configuration Aspect

The start register address, index register address, data buffer size, index correction and PLC time stamp register address mentioned below depend on how the dialed controller is programmed. For more information, see Dialed History on page 100.

4. From the History Log drop-down list, select the relevant dialed history direct log.

Make sure to select a **direct** log, and not any hierarchical log.

5. Enter the start register address in the Start register box. The start register is the first of the registers used in the dialed controller to store the historic values for the signal.
6. To use the default history settings, select the Use Default PLC Values check box, and then skip to the next step. Otherwise, clear it and then:
   – In the Index Register box, enter the index register address. The index register is the register used in the dialed controller to keep track of where the latest historic value for the signal is stored.
   – In the # Registers box, enter the number of registers used in the dialed controller to store the historic values for the signal. This number is also called the data buffer size.
   – In the Index Correction box, enter the index correction value. This value is used to offset where the latest historic value is stored. It is usually equal to 0 or 1.
   – Select either Use Computer Time Stamp or Use PLC Time Stamp. When historic data is collected, the time stamp is used to determine the time of the last sample.

     Computer time stamp: the time for the call to the controller is used.

     PLC time stamp: the time is read from two registers in the controller. Enter the address for the first of these two registers in the Address box.
7. Click **Apply**.
8. Repeat Step 1 to Step 6 for each other signal part of a dialed controller.
9. Deploy the configuration for each affected control network.
10. The dial history configuration is now finished.

INDEX

Contact us

ABB AB
Control Systems
Västerås, Sweden
Phone: +46 (0) 21 32 50 00
Fax: +46 (0) 21 13 78 45
E-Mail: processautomation@se.abb.com
www.abb.com/controlsystems

ABB Inc.
Control Systems
Wickliffe, Ohio, USA
Phone: +1 440 585 8500
Fax: +1 440 585 8756
E-Mail: industrialitsolutions@us.abb.com
www.abb.com/controlsystems

ABB Industry Pte Ltd
Control Systems
Singapore
Phone: +65 6776 5711
Fax: +65 6778 0222
E-Mail: processautomation@sg.abb.com
www.abb.com/controlsystems

ABB Automation GmbH
Control Systems
Mannheim, Germany
Phone: +49 1805 26 67 76
Fax: +49 1805 77 63 29
E-Mail: marketing.control-products@de.abb.com
www.abb.de/controlsystems

Copyright © 2003-2010 by ABB.
All Rights Reserved

3BSE035041-510

Power and productivity
for a better world™