

System 800xA Control

AC 800M

Configuration

System Version 6.0

Power and productivity
for a better world™



System 800xA Control

**AC 800M
Configuration**

System Version 6.0

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.

Products described or referenced in this document are designed to be connected, and to communicate information and data via a secure network. It is the sole responsibility of the system/product owner to provide and continuously ensure a secure connection between the product and the system network and/or any other networks that may be connected.

The system/product owners must establish and maintain appropriate measures, including, but not limited to, the installation of firewalls, application of authentication measures, encryption of data, installation of antivirus programs, and so on, to protect the system, its products and networks, against security breaches, unauthorized access, interference, intrusion, leakage, and/or theft of data or information.

ABB verifies the function of released products and updates. However system/product owners are ultimately responsible to ensure that any system update (including but not limited to code changes, configuration file changes, third-party software updates or patches, hardware change out, and so on) is compatible with the security measures implemented. The system/product owners must verify that the system and associated products function as expected in the environment they are deployed.

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/EC and in Low Voltage Directive 2006/95/EC.

TRADEMARKS

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

Copyright © 2003-2016 by ABB.
All rights reserved.

Release: April 2016
Document number: 3BSE035980-600 A

TABLE OF CONTENTS

About This User Manual

General	15
Document Conventions	16
Feature Pack	16
Warning, Caution, Information, and Tip Icons	17
Terminology	18
Related Documentation	18

Section 1 - Basic Functions and Components

Introduction	19
Control Project Templates	21
Control Projects	22
Program Organization Units, POUs	23
Entities and Reservation (Multi-User Engineering)	24
Entities	24
Reservation	25
Environments	27
Engineering and Production Environments	27
Remove Environment Changes	29
System Firmware Functions	30
Hardware	32
Standard System Libraries with Hardware	33
Customized Hardware Types	35
Configuring the Controller	36
Enabling Web Server	40
Basic Hardware	41

Basic Library for Applications.....	42
Application Types and Instances.....	43
Types and Instances - Concept.....	44
Define a Type in the Editor.....	45
Control Module Types, Function Block Types, and Diagram Types.....	54
Types in Applications.....	56
Types in User defined Library.....	57
Modify Complex Types.....	58
Diagram and Diagram Types.....	59
Decisions When Creating Types.....	64
Create and Connect Instances.....	66
Function Block Execution.....	71
Control Module Execution.....	73
Diagram Execution.....	74
Single Control Modules.....	76
FD Port.....	78
Aspect instances.....	80
Variables and Parameters.....	81
Variable and Parameter Concept.....	83
Variables.....	84
Variable Entry.....	85
Specific Initial Values.....	94
External Variables.....	96
Access Variables.....	97
Communication between Applications Using Access Variables.....	99
Communication in an Application Using Global Variables.....	100
Communication Variables.....	101
Control the Execution of Individual Objects.....	113
Link Variables in Diagrams.....	116
Project Constants.....	116
I/O Addressing Guidelines.....	121
Connecting Variables to I/O Channels.....	122

Extensible Parameters in Function Blocks.....	127
Keywords for Parameter Descriptions.....	128
Real value in AC 800M.....	129
Property Permissions.....	133
Property Attribute Override.....	134
Library Management	135
Connect Libraries	136
Import/Export Libraries	141
Create Libraries.....	141
Library States	142
Library Versions	143
Library Password Protection	146
Add Types to Libraries Used in Applications	147
Add Customized Hardware Types to Library.....	150
Device Import Wizard	152
Additional Files for Libraries with Hardware	153
Delete Hardware Types	156
Type Usage for Hardware Types	156
Hide and Protect Control Module Types, Function Block Types, Diagram Types, and Data Types	158
Protect a Self-Defined Type	159
Protect MySupervision Type Example.....	161
Task Control	165
Task Connections	165
Task Execution	169
Task Priority	169
Interval Time	172
Offset	173
Execution Time	178
Overrun and Latency	178
Overrun Supervision	179
Latency Supervision.....	181

Task Abortion.....	183
Load Balancing.....	184
Non-Cyclic Execution in Debug Mode.....	186
Task Analysis.....	187
Exploring the Interface.....	188
Modifying Task Execution Time.....	192
Error and Warning Categories.....	192
Security.....	195
Authentication at Download.....	196
Confirmed Online Write.....	197
Search and Navigation.....	198
Search and Navigation Dialog.....	198
Search Settings.....	199
Symbol and Definition.....	201
References.....	203
Navigation to Editors.....	208
Search and Navigation Settings.....	208
Search Data.....	212
Reports.....	212
Analog Input and Output Signal Handling.....	213
Backup Media.....	216
Card Types.....	217
Adding CF Card or SD Card to Hardware.....	218
Dump of Post Mortem Memory Image.....	219
Saving Cold Retain Values on Files.....	222
Downloading the Application to Removable Media.....	224
Configuration Load.....	224
Upgrading Controller Firmware using Backup Media.....	225
Controller Restart Modes and Backup Media Usage.....	231
Storing Related Files.....	232
Restoring Formatted CF Cards to Original Size.....	232
Remove Files Completely from a CompactFlash Card.....	233

Compiler Switches.....	233
Settings	233
Reports.....	237
Difference Report.....	237
Difference Report Viewer	244
Source Code Report	245
Reports Generated at Download	248
Portability Verification	251
Performance Management.....	251
Project Documentation	253
Objects and Types	255
Editor Items.....	255
Used Types	256

Section 2 - Alarm and Event Handling

Introduction	257
Alarms and Events	258
Alarm and Event Library.....	259
Process Alarm and Event Generation.....	259
Process Alarms and Events	260
Detection of Simple Events.....	269
Built-in Alarm and Event Handling in Other Libraries.....	269
External Time Stamps (S800 I/O).....	274
External Time Stamps IEC 61850.....	274
External Time Stamps (PROFINET IO)	275
External Time Stamps (INSUM).....	276
Choose Alarm Handling Method for INSUM Alarms.....	281
System Alarm and Event Generation	282
Controller Generated System Alarms and System Simple Events.....	283
User Generated System Alarms	285
Handling Alarms and Events.....	285
Simple Events.....	286
System Alarms and Events.....	286

Time Stamps	286
Alarm and Event Communication.....	289
Subscriptions.....	289
Configuration of OPC AE Communication – Overview	289
Buffer Queues	291
Buffer Configuration.....	292
Local Printers	293
Print Format	293
Sending an Alarm to the Application	295
Third Party OPC Clients	296
Translation – NLS Handling of Strings.....	296
Alarm Examples.....	297
AlarmSimple_M Example	298
Alarm and Event Aspect Example (AlarmSimple_M)	303
Alarm Owner Examples.....	304
Condition State Example	308
Inhibit Example.....	310
Simple Event Examples	312
Alarm and Event Functions.....	316
System Diagnostics.....	316
Acknowledgement Rules – State Diagrams.....	317
Alarm Shelving	321
Section 3 - Communication	
Introduction	323
Communication Libraries.....	324
COMLI Communication Library	324
Foundation FIELDBUS HSE Communication Library	324
INSUM Communication Library	328
MB300 Communication Library	332
MMS Communication Library	333
MODBUS RTU Communication Library	334
MODBUS TCP Communication Library	334

Modem Communication Library.....	334
Siemens S3964 Communication Library	334
SattBus Communication Library.....	335
MTM Communication Library	335
Serial Communication Library.....	336
TCP and UDP Communication Libraries	338
Generic IO Communication Library	339
Supported Protocols.....	340
Control Network.....	341
Network Redundancy	341
Statistics and Information on Communication.....	342
Variable Communication	343
Access Variables.....	343
Communication Variables	343
StartAddr	345
Reading/Sending Data	348
Connection Methods	350
Communication Concepts	352
Fieldbus Communication.....	355
HART Communication.....	358
SIL Certified Communication	358
SIL Communication Using IAC.....	359
SIL Communication using MMSCommLib.....	362
Section 4 - Online Functions	
Introduction	367
Online Editors.....	368
Diagram Editor in Test Mode and Online Mode.....	370
Dynamic Display of I/O Channels and Forcing	372
Forcing I/O Channels in SIL Applications.....	374
Scaling Analog Signals.....	375
Supervising Unit Status	375
Find Out What is Wrong by Using HWStatus	376

AllUnitStatus	377
Binary Channels	378
Supervising Communication Variable Status	379
Supervising Communication Variable Using: Status Notation	379
Supervising Communication Variable Using GetCVStatus	380
Understanding the Complete Status Code	383
Status Indications	386
Acknowledge Errors and Warnings	387
Tasks	387
Interaction Windows	388
Status and Error Messages	390
Search and Navigation in Online and Test Mode	391
Project Documentation	395

Section 5 - Maintenance and Trouble-Shooting

Introduction	397
Running Control Builder on Terminal Server	398
Characteristics of Control Builder as Terminal Server	398
Backup and Restore	401
Introduction	401
Files for Separate Backup	402
Remove and Add FSD Server Files	402
Compiler Output File Helper	402
Migration	409
Migration from 800xA to Compact Control Builder	409
Migration from Compact Control Builder to 800xA	411
Import and Export	413
Introduction	413
Export a Library	413
Export an Application/Controller	415
Import an Application/Controller	415
Import and Export Alternatives	416
Applying Cold Retain Values when Importing Applications	417

About Library Import/Export.....	418
Detailed Difference Report During Import	419
Start Values Analyser	419
Controller Configuration	422
Controller Settings in Non-High Integrity Controllers	424
Controller Settings in High Integrity Controllers.....	427
Error Handler Log Entries.....	430
Online Upgrade	431
Plan for Online Upgrade already at Project Design Phase.....	431
Why You Need to Read this First.....	433
Restrictions for Online Upgrade	434
Preliminary Actions for Online Upgrade	435
Online Upgrade Process	439
Running Online Upgrade	444
Solving an Interrupted Online Upgrade	445
Trouble-Shooting	446
General	446
Log Files	447
Crash Dumps for Analysis and Fault-Localization	467
Remote Systems Information.....	468
Diagnostics for Communication Variables.....	471
Analysis Tools	477
System Diagnostics	479
Trouble-Shooting Error Symptoms.....	485
Common Reason for Shut-Down AC 800M HI Controller.....	488
Connection to Aspect Server.....	492
Error Reports	493
 Appendix A - Array, Queue and Conversion Examples	
Arrays	495
SearchStructComponent.....	497
InsertArray	501
SearchArray.....	502

Queues.....	506
Conversion Functions.....	510
DIntToBCD.....	510
BCDToDInt.....	511
ASCII.....	512
ASCII Conversion.....	514

Appendix B - System Alarms and Events

General.....	521
OPC Server – Software.....	522
OPC Server – Subscription.....	524
Controller – Software.....	526
Controller – Hardware.....	558
Alarms and Events Common for all Units.....	560
Unit Specific Alarm and Events.....	564

INDEX

Introduction.....	577
Revision History.....	577
Updates in Revision Index A.....	577

About This User Manual

General



Any security measures described in this User Manual, for example, for user access, password security, network security, firewalls, virus protection, etc., represent possible steps that a user of an 800xA System may want to consider based on a risk assessment for a particular application and installation. This risk assessment, as well as the proper implementation, configuration, installation, operation, administration, and maintenance of all relevant security related equipment, software, and procedures, are the responsibility of the user of the 800xA System.

This user manual describes how to use the basic 800xA programming and configuration functions that can be accessed via the Plant explorer and Project Explorer interfaces.

The libraries described in this manual conform to the IEC 61131-3 Programming Languages standard, except for control modules and diagrams, which are not supported by this standard.

- [Section 1, Basic Functions and Components](#), describes all the basic functions that are available via system functions, Basic library, and commands in the Control Builder interface. This section also describes the type and object concept, and how variables and parameters are used.
- [Section 2, Alarm and Event Handling](#), describes the types in the Alarm and Event library and how to use them to add alarm and event functions to objects that do not have alarm functionality built into them.
- [Section 3, Communication](#), describes the types in the Communication libraries and how to use them to establish communication between controllers.
- [Section 4, Online Functions](#), describes Control Builder functions in online mode.

- [Section 5, Maintenance and Trouble-Shooting](#), describes Control Builder maintenance functions. It also describes how to use the Import/Export function, how to write an error report, the location of various log files, how to read these log files, and how to fix some common problems.
- [Appendix A, Array, Queue and Conversion Examples](#) contains some examples on how to use queues and arrays, and how to convert numbers from one format to another.
- [Appendix B, System Alarms and Events](#) describes system alarms and system simple events from a controller perspective.



Before running SIL certified applications in a High Integrity controller, refer to *System 800xA Control AC 800M Getting Started (3BSE041880*)* manual.

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, etc.

Feature Pack

The Feature Pack content (including text, tables, and figures) included in this User Manual is distinguished from the existing content using the following two separators:

[Feature Pack Functionality](#)

<Feature Pack Content>

Feature Pack functionality included in an existing table is indicated using a table footnote (*):

*[Feature Pack Functionality](#)

Feature Pack functionality in an existing figure is indicated using callouts.

Unless noted, all other information in this User Manual applies to 800xA Systems with or without a Feature Pack installed.

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 the 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 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*.

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 the 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 Basic Functions and Components

Introduction

Control Builder is a programming tool that contains:

- Compiler
- Graphical programming editors that provide graphical representation of the whole logic.
- Programming editors for IEC-61131 languages
- Standard libraries for developing controller applications
- Standard hardware types (units) in libraries for configuring the controller

The Control Builder tool also includes system firmware and common functions such as control system templates, task supervision, security and access management. Most of the application development can be accomplished using the basic functions and components presented in this section.

This section is organized in the following manner:

- [Control Project Templates](#) on page 21 describes the different templates that can be used to create a control project.
- [Control Projects](#) on page 22 describes how to create and work with control projects.
- [Program Organization Units, POU](#) on page 23 introduces the Program Organization Unit (POU) concept.
- [Entities and Reservation \(Multi-User Engineering\)](#) on page 24 introduces the concept of reservation and entities.
- [Environments](#) on page 27 introduces the concept of environments.

- [System Firmware Functions](#) on page 30 describes firmware functions included in the system, which can be used in any application.
- [Hardware](#) on page 32 describes the standard libraries for hardware types.
- [Basic Library for Applications](#) on page 42 describes the objects of the Basic library, which can be included in any project.
- [Application Types and Instances](#) on page 43 introduces the very important, object-oriented, types and objects concept. This subsection also describes how to add user defined types and how to create objects (instances) from types.
- [Variables and Parameters](#) on page 81 describes how to use parameters and variables to store and transfer values in the control system.
- [Library Management](#) on page 135 describes how to work with libraries.
- [Hide and Protect Control Module Types, Function Block Types, Diagram Types, and Data Types](#) on page 158 describes how to hide and protect objects and types, using the Hidden and Protected attributes.
- [Task Control](#) on page 165 describes how to set up tasks to control the execution of the applications.
- [Overrun and Latency](#) on page 178 describes how to configure overrun and latency control for the tasks.
- [Task Analysis](#) on page 187 describes the Task Analysis tool that detects the possible task overrun/latency problems in an application before its download to the controller.
- [Security](#) on page 195 describes how to set up access to actions and objects, as well as how to set up access rights for SIL certified applications.
- [Search and Navigation](#) on page 198 describes how to use the search and navigation function to find all instances of a type or to find out where a certain variable is used.
- [Analog Input and Output Signal Handling](#) on page 213 describes how to enable over and under range for input and output objects.
- [Backup Media](#) on page 216 describes how to use the Backup Media as a removable storage.

- [Compiler Switches](#) on page 233 describes how to use Compiler Switches to control the behavior of compiler.
- [Reports](#) on page 237 describes the function of the Difference Report and Source Code Report.
- [Performance Management](#) on page 251 describes how to gather information of the applications using the Compiler Statistics tool.
- [Project Documentation](#) on page 253 describes how to use the Project Documentation function to document standard libraries, user defined libraries, and applications in MS Word format.

Control Project Templates

A control project template sets up the necessary features required to build a control project. The control project consists of system firmware functions, basic library functions, application functions and a pre-set of hardware functions.

The 800xA System provides the following AC 800M control project templates:

- **AC800M**
Template for normal use, and for running non-SIL applications.
- **AC800M_HighIntegrity_SM811**
Template for running non-SIL, SIL1-2, and SIL3 applications.
- **AC800M_HighIntegrity_SM812**
Template for running non-SIL, SIL1-2, and SIL3 applications.
- **EmptyProject**
Template that requires a minimum configuration, with only the System folder inserted. This template is rarely used.
- **SoftController**
Template for developing software for simulating non-SIL applications without a controller.
- **SoftController_HI**
Template for developing software for simulating SIL applications without a controller.

For example, the AC 800M_HighIntegrity_SM811 template prepares a control project for a PM865 CPU and an SM811 module, while the AC 800M template and the SoftController template have completely different settings. The EmptyProject template contains only the compulsory system firmware functions, with no additional application or hardware functions.

A control project template can be selected from a dialog, when creating a control project. For more information about creating a control project, see [Create and Open a Control Project in Plant Explorer](#) on page 22.

Control Projects

A control project combines the control applications and the controllers together in the Project Explorer. Several control projects can be created for the same control network.

The control projects can be created either from the Plant Explorer or from the Project Explorer.



Before creating a control project, set up and configure a control network in the Control Structure (Plant Explorer).

Create and Open a Control Project in Plant Explorer

1. In the Control Structure, right-click **Control Network** and select **New Object** to open the New Object window.
2. Select a control project template and enter a name for the control project in the **Name** field.
3. Click **Create** to create a new control project.

The 800xA system starts the Control Builder, and the control project opens in Project Explorer.



It is not required to close the Control Builder each time when a new control project is to be opened. Control Builder automatically closes the previous project and opens the new project in the background.

A SIL application can only run in an AC 800M High Integrity (HI) controller. Create SIL applications by selecting the High Integrity control project template (AC800M_HighIntegrity). See [Control Project Templates](#) on page 21. A control

project containing a VMT library, a VMT application, and a CTA application is obtained if this template is used.



The VMT library, VMT application, and CTA application are created to check that the High Integrity controller and the compiler work properly. These libraries and the compiler test application are used for internal checks only. Do not try to alter or remove these applications or the VMT library.

For more information, refer to SIL Certified Applications in the manual *System 800xA Control AC 800M Getting Started (3BSE041880*)*. Also, refer to *System 800xA Safety AC 800M High Integrity Safety Manual (3BNP004865*)*, which contains guidelines and safety considerations for all safety life cycle phases of an AC 800M High Integrity controller.

Program Organization Units, POU

The IEC 61131-3 standard describes programs, function blocks, and functions as Program Organization Units (POUs). The Control Builder also considers control modules and diagrams as POU. All these units are helpful in organizing the control project into code blocks, minimizing code writing, and optimizing the code structure and code maintenance.

A POU is an object type that contains an editor to write code and declare parameters and variables.

All POU. s can be repeatedly used in a hierarchical structure, except for diagrams and programs that can only be a 'top-level' POU, inside an application.

Entities and Reservation (Multi-User Engineering)

Entities and reservation provide support for multi-user engineering (working within a project development group that involves several people).

Before modifying the properties of an object, the object must be reserved. This ensures that only one user can modify an object at a time. This also protects configuration data from being unintentionally modified when multiple users are working on one system.



Reservations do not protect any runtime data or prevent download of modified applications to a controller. For example, if a controller is reserved by user *A*, and an application is reserved by user *B*, it is still possible for user *C* to download the application. However, reservations are indicated in the Download dialog.

A single user who has logged on to more than one client, and several users who use the same user account, can unintentionally overwrite configuration data.

If a user releases the reservation on an object, another user can reserve and modify the object. However, it is only possible to make a reservation of entities, that is, the smallest subset of objects that can be reserved is an entity.

An entity is a set of objects (with aspects) that is reserved as a single unit.

Unless an entity is reserved, parts of the Project Explorer will be read-only. For example, some context menu items are disabled, and dialog boxes are read-only.



If environments are used, and a user reserves an entity in one environment, another user can reserve the same entity in another environment.

Entities

The following objects are grouped as entities:


- Projects, applications, controllers
- Libraries, libraries with hardware types
- Control modules types, except hidden control module types
- Function block types, except hidden function block types
- Diagram types, except hidden diagram types
- Diagrams

An entity can be part of another entity. For example, applications and controllers are part of a project, and control module types, function block types, and diagram types are part of either an application or a library.

When an entity is reserved, all its objects are reserved. For example:

- When the user reserves a controller, all objects that are part of the controller (objects such as hardware units and tasks) are reserved.
- When the user reserves an application, its programs and data types, but not necessarily its diagrams, function block types or control module types, are reserved.



In case environments are used, the entity icons in Project Explorer show only the reservation status for the existing environment. For example, the  icon is shown for the current environment; however, the Reservation dialog shows complete reservation status.

Reservation

The entity must be reserved before it can be modified.

To reserve an entity:

1. Right-click the entity (for example, an application), and select **Reserve** to open the Reserve dialog box.
2. Select the entities to reserve. Click **Help** for more information on how to use the dialog box.





The same dialog box (with a different name) also appears when an operation that requires the reservation of one or more entities is performed.

To release the reservation of an entity after modifying it:

1. Right-click the entity (for example, an application), and select **Release Reservation** to open the Release Reservation dialog box.
2. Select the reservations to release. Click **Help** for more information on how to use the dialog box.



In case environments are used, the reservation can only be released for the current environment.

Use the Reserve  and Release Reservation  icons in the Project Explorer toolbar to reserve entities or to release the reservation. Some offline editors also have a Reserve button.



To take over a reservation, both the Plant Explorer and the Project Explorer can be used. For more information, refer to the *System 800xA Configuration (3BDS011222*)*.

Environments

In 800xA Systems, environments provide isolated engineering. Since different environments can have different content, a control application can be modified without affecting the running control application. For example, the Engineering Environment can contain a modified application, while the Production Environment contains the running application.



Environments require a separate license and are not available to all users. The Project Explorer shows the information about an environment only when it is being used.



For more information, refer to the *System 800xA Engineering Engineering and Production Environments (3BSE045030*)*.

Engineering and Production Environments

When environments are used, the basic combination is to have one Engineering Environment and one Production Environment:

- Engineering Environment is used for engineering (For example, to modify a project or an application).
- Production Environment is used to download a project (or a single application) to the controller and go online. An operator can then use an Operator Workplace opened in this environment to control the process.

When an entity in an environment is modified, the changes are visible in that environment only, and not in any other environment. All users working in the same environment can see the changes made by each other.

The user can transfer the modified entities from one environment to another. This is called Deploy.

When a modified application is deployed from the Engineering Environment to the Production Environment, the Production Environment no longer contains the running application. Instead, the Production Environment contains the modified application, which can be downloaded to the controller.



To change to another environment in the Control Builder, re-open the project in the relevant environment.



When a project is deployed from the Engineering Environment to the Production Environment, there is a possibility that a new application was created only in the Production Environment and not in the Engineering Environment. In this case, this application is not deleted from the Production Environment.

Environment Workflows

For a new project, follow this workflow:

1. Create a new project in the Engineering Environment, and modify the entities as desired.
2. Deploy the project and all other modified entities from the Engineering Environment to the Production Environment.
3. Re-open the project in the Production Environment and download the new project to the controller.

To modify an existing project, follow this workflow:

1. Open the project in the Engineering Environment.
2. Right-click the project name, and select **Refresh Project**.
3. Modify the project without affecting the Production Environment, which contains the project running in the controller.
4. Deploy the modified project to the Production Environment.
5. Re-open the project in the Production Environment, and download the modified project to the controller.

Deploying an Entity



Deploy is only available in offline mode.

To deploy an entity (for example, an application):

1. Right-click the entity, and select **Deploy**.
2. Use the displayed Deploy dialog box to deploy the entity to the desired environment.



The Deploy dialog box is the same as in Plant Explorer. For more information on how to use the dialog box, click the **Help** button or refer to the *System 800xA Configuration (3BDS011222*)*.

Remove Environment Changes

When a project is opened in the Engineering Environment, the project may already contain changes.

To start working with the same project as in the Production Environment:

Either

- Refresh the Engineering Environment, which recreates the entire Engineering Environment as a copy of Production environment.

Or

- Replace selected entities.

In Engineering Environment, single entities can be selected and updated to be identical with the Production environment. Refer to the manual *System 800xA Engineering, Engineering and Production Environments (3BSE045030*)* for more details.

System Firmware Functions

All system firmware functions are stored in the System folder, which is located at the top of the library branch (in Project Explorer).



The System folder is not a library, even though it is always shown in the library branch, together with the libraries (Basic library, Icon library, etc.)

The System folder contains fundamental IEC 61131-3 data types and functions, along with other firmware functions, which can be used in firmware in the controller. They are all protected and automatically inserted via the selected control system templates.

The System folder cannot be changed, version handled or deleted from a control project.

The system firmware functions that can be used in the application depends on the Firmware version. To upgrade the Firmware, replace the BasicHWLib with the latest version.

[Table 1](#) contains the System firmware data types and functions. Refer the Control Builder online help for more information and description.



To access the detailed online help and how-to-do instructions for a system firmware function, select the data type or function, and press the F1 key.

Table 1. System Function Overview

System Functions	Examples
Simple Data Types	bool, dint, int, uint, dword, word, real, etc.
Structured Data Types	time, Timer, date_and_time, etc.
Common Library Data Types	Open structured data types like, BoolIO, DintIO, DwordIO, RealIO, HWStatus, SignalPar, etc.
Bit String Operations	and, or, xor, etc.
Relational and Equality Functions	Equal to, Greater than, etc.
Mathematical Functions	Trigonometric, Logarithmic, Exponential and Arithmetic Functions.
Data Type Conversion	Conversion of bool, dint, etc.
String Functions	Handles strings like, inserts string into string, deletes part of a string, etc.
Exception Handling	Functions for handling zero division detection integer and real values.
Task Functions	SetPriority, GetPriority, etc.,. Handles the priority of the current task.
System Time Functions	Exchanging time information between different systems.
Timer Functions	Functions to Start, Stop and Hold Timers.
Random Generation Functions	Functions for generating random numbers or values.
Variable Handling Functions	Reads and writes variable values. Provides status information of communication variables.

Table 1. System Function Overview (Continued)

System Functions	Examples
Array Functions	Handles arrays.
Queue Functions	Handles queues.

Hardware

All hardware is defined as hardware types (units) in Control Builder. The hardware types reflect the physical hardware in the system.

Hardware types are organized and installed as libraries. This makes it possible to handle hardware types independently, with the following advantages:

- Since the libraries are version handled, different versions of the same hardware type exist in different versions of the library. This makes it easy to upgrade to newer system versions and also allows coexistence of new and old hardware units.
- The new versions of a library (along with the hardware types) can be easily delivered and inserted to the system.

A number of standard libraries with hardware types are delivered with the system. A standard library is write protected and cannot be changed

- Only used hardware types allocate memory in the controller.

Standard System Libraries with Hardware

The standard system libraries with hardware are delivered by the system. [Table 2](#) describes the standard libraries with hardware.

Table 2. Standard system libraries with hardware

Library	Description
ABBDrvFenaCI871HwLib	Fieldbus adapter for ABB drives, F-series through the PROFINET IO
ABBDrvRetaCI871HwLib	Fieldbus adapter for ABB drives, R-series through the PROFINET IO
ABBDrvFpbaCI854HwLib	Fieldbus adapter for ABB drives, F-series through the PROFIBUS DP
ABBDrvRpbaCI854HwLib	Fieldbus adapter for ABB drives, R-series through the PROFIBUS DP
ABBDrvNpbaCI854HwLib	Fieldbus adapter for ABB drives, N-series through the PROFIBUS DP
ABBPnQ22CI871HwLib	Ethernet adapter to connect up to four FBP FieldBusPlug devices like UMC100 or PST through the PROFINET IO
ABBMNSiSCI871HWLIB	Motor control center solution that can be used in PROFINET IO network.
ABBPnI800CI854HwLib	ABB Panel 800 for PROFIBUS
ABBPnProcCI854HwLib	ABB Process Panel for PROFIBUS
BasicHWLib	Basic controller hardware types for AC 800M and SoftController
BasicHIHwLib	Basic controller hardware types for AC 800M HI and SoftController HI
CI853SerialComHwLib	RS-232C serial communication interface
CI854PROFIBUSHwLib	Communication interface PROFIBUS DP-V1
CI855Mb300HwLib	Communication interface MasterBus 300

Table 2. Standard system libraries with hardware

Library	Description
CI856S100HwLib	Communication interface S100 I/O system and S100 I/O units
CI857InsumHwLib	Communication interface INSUM
CI858DriveBusHwLib	Communication interface DriveBus
CI860FFhseHwLib	Communication interface FOUNDATION Fieldbus HSE
CI862TRIOHwlib	Communication interface for TRIO
CI865SattIOHwLib	Communication interface for remote I/O connected via ControlNet
CI867ModbusTcpHwLib	Communication interface MODBUS TCP
CI868IEC61850HwLib	Communication interface IEC 61850
CI869AF100HwLib	Communication interface for AF 100
CI871PROFINETHwLib	Communication interface CI871
CI872MTMHwLib	Communication interface for MOD5-to-MOD5
CI873EthernetIPHWLib	Communication interface EtherNet/IP
ModemHwLib	Modem unit
PrinterHwLib	Printer unit
S200IoCI854HwLib	S200 adapter and S200 I/O units for PROFIBUS
S200IoCI873HwLib	S200 slave and I/O units for EthernetIP (CI873).
S800CI830CI854HwLib S800CI840CI854HwLib S800CI801CI854HwLib	S800 adapters and S800 I/O units for PROFIBUS
S800IoModulebusHwLib	S800 I/O units for ModuleBus

Table 2. Standard system libraries with hardware

Library	Description
S900IoCI854HwLib	S900 adapter and S900 I/O units for PROFIBUS
SerialHwLib COMLIHWLib ModBusHWLib S3964HWLib TCPHwLib UDPHwLib	Hardware libraries for direct controller communication with external hardware, using different communication protocols



For a complete list of the hardware types in the standard libraries, see Control Builder online help.



If a suitable hardware type cannot be found in any of the standard system libraries, it can be found in the Device Integration Library. The Device Integration Library can be purchased separately from ABB.

Customized Hardware Types

Customized hardware types can be created in user-defined libraries, using the Device Import Wizard. This is useful when the hardware types found in the standard system libraries or the Device Integration Library are not sufficient.

The Device Import Wizard imports a device capability description file (for example, a *.gsd file), converts the file to a hardware type, and inserts the type into the user-defined library (See [Create Libraries](#) on page 141. Also see [Device Import Wizard](#) on page 152 and [Supported Device Capability Description Files](#) on page 152).

User-defined libraries with hardware types are included while performing import and export, and backup and restore, in the Plant Explorer. By using the import and export function, it is possible to distribute the user-defined libraries with hardware types. These libraries are developed centrally, or by ABB for a specific project to other engineering systems. For further information, see [Import/Export Libraries](#) on page 141.

In exceptional cases, it may be relevant to insert individual external customized hardware types to a user-defined library (for example, to use a specific hardware type, which have been converted and used in an earlier version of Control Builder).

The Source Code Report can be used to view the hardware types loaded in the project. See [Source Code Report](#) on page 245.

Configuring the Controller

Before configuring the controller:

1. Insert the libraries, which contain the hardware types (units) to be used in the controller configuration, into the control project.
2. Connect the libraries to the controller.

See [Connect Libraries](#) on page 136 for information on how to insert and connect libraries.

Add Unit to Hardware in Controller Configuration

Perform the following steps to add a new hardware unit into the controller configuration in Project Explorer:

1. Make sure that the library, which contains the hardware type to be added, is inserted to the project and connected to the controller.
2. Right-click the unit to which a new hardware unit is to be added, and select **Insert Unit** to open the Insert Unit dialog.

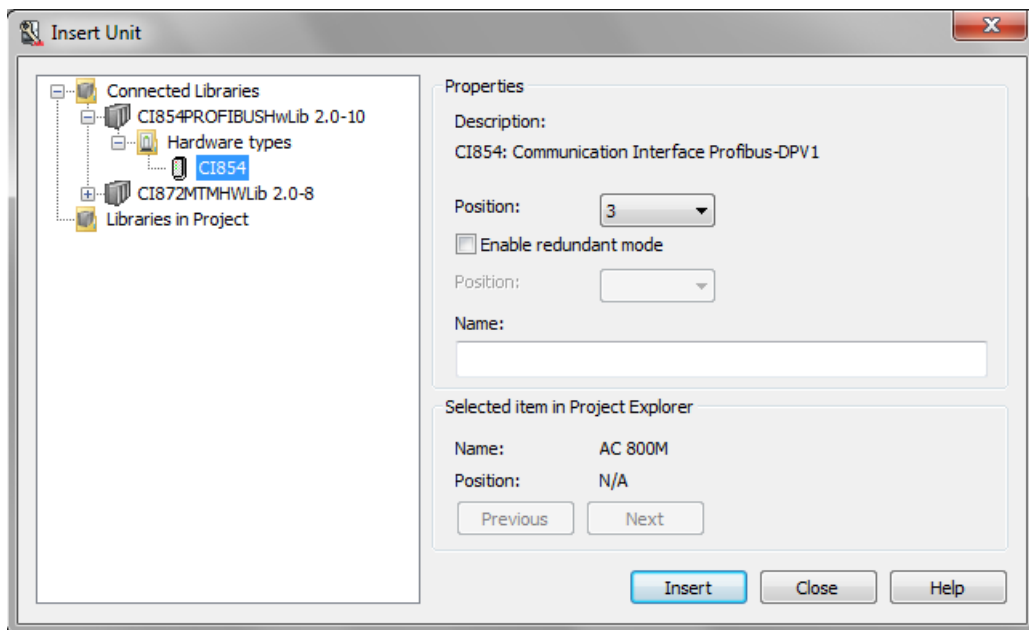


Figure 1. Insert Unit dialog for inserting hardware in a controller configuration



It is not possible to select **Insert Unit** if the unit cannot contain any sub-units or if no more positions are available.

- Expand the relevant library folder under **Connected Libraries**, and select the hardware type to be included.



The **Libraries in Project** contains libraries that are added to the project but not yet connected to the controller. If a unit is selected under **Libraries in Project**, the option to connect the library to the controller appears.

- From the **Position** drop-down list, select a position for the hardware unit.
By default, the first available position is chosen. If no more positions are available, the Position drop-down list is empty and the **Insert** button is disabled.

- For units supporting redundancy, check the **Enable redundant mode** check box, and select a position for the backup unit.



Some redundant units have a fixed position offset. For these units, the backup position is automatically calculated, and the user cannot change this position.



Click **Previous** or **Next** to navigate to another unit in the Project Explorer hardware tree.

- In the **Name** field, enter a name for the unit. After the unit is inserted in the hardware tree, this name appears along with the name of the selected type.
- Click **Insert** to apply the changes made.
- Click **Close** to close the dialog.



To rename the unit after it is inserted, right-click the unit, and select **Rename Unit**.

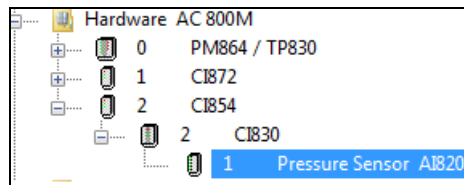


Figure 2. Example of a hardware tree with a name for the AI820 unit

Replace Hardware in a Controller Configuration

Perform the following steps to replace a hardware unit in a controller configuration:

- Make sure that the library, which contains the hardware type to be added, is inserted to the project and connected to the controller.
- Right-click on the unit to be replaced, and select **Replace Unit** to open the Replace Unit dialog.



The Replace Unit dialog works in the same way as the Insert Unit dialog, except that it is not possible to change the position of the unit in the Replace Unit dialog.

While the hardware unit is being replaced in a controller configuration, the system retains the settings and connections, and also retains the units in the existing subtrees. For example, replacing a CPU with a similar one can be done without any connection loss or data loss.

Enabling Web Server

The diagnostics and maintenance operations of Communication Interface modules are done using the web server feature. By default web server is disabled and secured.

The web server for Communication Interface modules can be accessed and enabled on request using the **Enable Web Server** option from the control Builder. This option is visible only when Communication Interface modules are configured and the controller is online.

To enable the web server, right click the **PM8xx** controller module and select **Enable Web Server** option.

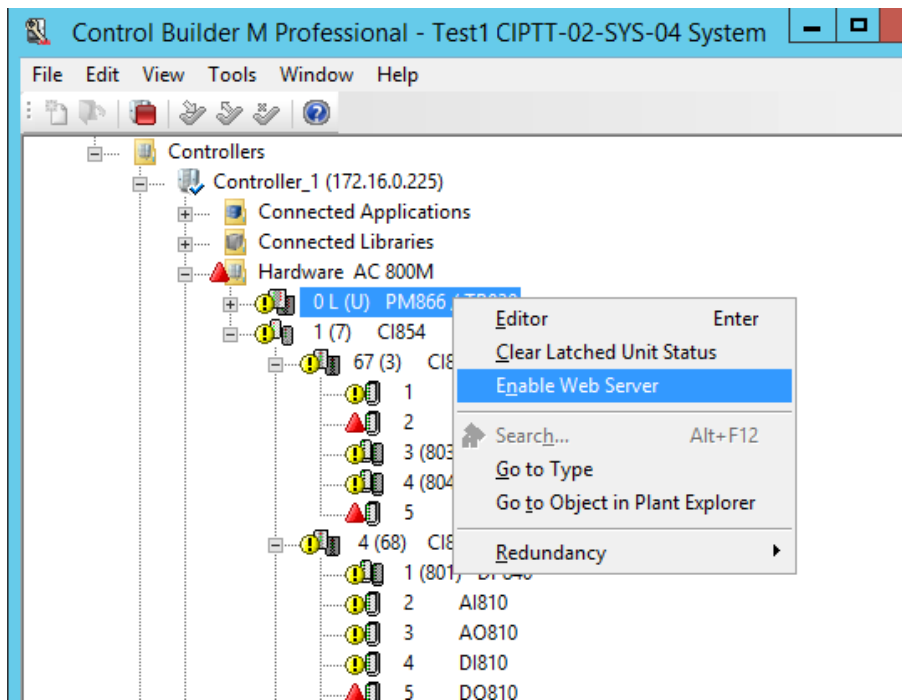


Figure 3. Enabling Web Server

For more details on diagnostics and maintenance operations of CI modules using web server, refer to the respective CI manuals, *AC 800M PROFINET IO*

Configuration (3BDS021515), AC 800M PROFIBUS DP Configuration (3BDS009030*), AC 800M FOUNDATION Fieldbus HSE (3BDD012903*).*

Basic Hardware

The two Basic Hardware Libraries: BasicHwLib and BasicHIHwLib, contain standard system hardware types that are used when configuring the AC 800M controller and SoftController. The standard system hardware types are installed along with the Control Builder.



Only one version of a Basic Hardware Library can be connected to a controller.

The BasicHwLib contains the following basic controller hardware:

- Controllers (AC 800M and SoftController)
- Compact Flash (CF) units
- Secure Digital (SD) units
- CPU units (PM8xx and CPU)
- Ethernet links, serial Com ports, and PPP ports
- ModuleBus
- IP
- IAC MMS

The BasicHIHwLib contains the following basic controller hardware:

- Controllers (AC 800M HI and SoftController HI)
- CPU unit (PM865 HI)
- CPU unit (PM867)
- SM811 and SM812 units
- Ethernet links, serial Com ports, and PPP ports
- ModuleBus
- IP
- IAC MMS

Basic Library for Applications

The Basic library contains basic building blocks for AC 800M control software. It contains data types, function block types and control module types with extended functionality, designed by ABB.

The contents inside the Basic library can be categorized as follows:

- IEC 61131-3 Function Block Types.
- Other Function Block Types.
- Control Module Types.



For a complete list of data types, function block types, and control module types in the Control Builder standard libraries, refer to the manual *System 800xA Control AC 800M Configuration (3BSE035980*)*

Table 3. Basic Library Overview

Basic Functions	Examples
IEC 61131-3 Function Block Types	Standard bistable function block types (SR, RS). Standard edge detection function block types (R_TRIG, F_TRIG). Standard counter function block types (CTU, CTD, etc.) Standard timer function blocks type (TP, TOn, etc.)
Other Function Block Types	ACOF (Automatic Check Of Feedback) functions, converters, pulse generators, detectors, system diagnostics, timers, compares, etc.
Control Module Types	Connection module for group start sequences (GroupStartObjectConn), ControlConnection inputs and outputs, Error Handler, Forced Signals, and acknowledgment of ISP values for communication variables.

Application Types and Instances

Types and instances form the basis of the application structure. This subsection contains an overview of the following:

- The type and instances concept, see [Types and Instances - Concept](#) on page 44.
- The editors that are used to create and configure the types, see [Define a Type in the Editor](#) on page 45.
- Important differences between control module types, function block types, and diagram types, see [Control Module Types, Function Block Types, and Diagram Types](#) on page 54.
- How to create types directly in an application, and how to create types in the library for re-use in applications. See [Types in Applications](#) on page 56 and [Types in User defined Library](#) on page 57.
- How to create complex types so that they are flexible enough for future upgrades, see [Modify Complex Types](#) on page 58.
- What to consider and what to set up before creating types and using them, see [Decisions When Creating Types](#) on page 64.
- How to create objects from types and connect the object to the surrounding application or type, see [Create and Connect Instances](#) on page 66.
- Details about diagrams and diagram types, see [Diagram and Diagram Types](#) on page 59.
- How different objects are executed, see [Function Block Execution](#) on page 71, [Control Module Execution](#) on page 73, and [Diagram Execution](#) on page 74.
- How to use single control modules as containers for control modules, see [Single Control Modules](#) on page 76.
- The aspect object setting, see [Aspect instances](#) on page 80.

Types and Instances - Concept

Types are used to represent motors, valves, tanks, etc. that are located in a plant area, and then turn them into manageable units in a control project (for example, motor types, valve types, mixer types, and so on). Instances are created based on each of these types.

A type is the source (the blue print) for a unit (motor, valve, tank, etc), while an instance represents the unit(s) in libraries and applications. There is an inherited mechanism between a type and all its instances, where all instances have the same performance as the type, and changes performed in the type affect all instances simultaneously.

A type is a generic solution, which can be used by many instances, and contains programming code with variables, functions, connection parameters (textual and graphical), graphical instances, and formal instances¹.

Figure 4 shows the relationship between a type located in a library and two instances created in an application.

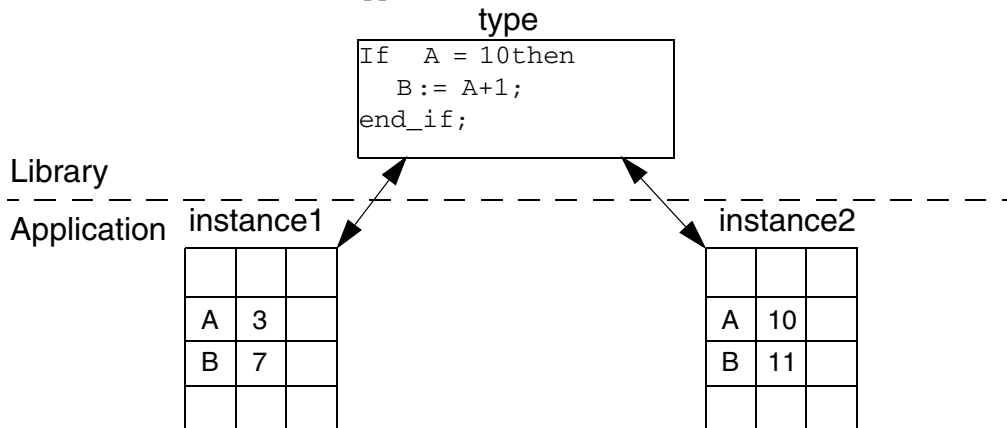


Figure 4. Relationship between a type and two instances.

The type contains the code, whereas each instance contains a list of computed variable values. The instance does not contain any code; it uses the code inside the type for manipulating its own local variable values.

1. Formal instances are instances of another type located inside a type. These, along with instances based on that type are executed in applications.

A type is always static and cannot run by itself in applications. To execute the code inside the type, an instance based on the type must be created. The instance executes the code located inside the type. To create an instance, point to a type either in a library or in an application.

All instances based on the same type have the same characteristics, which means they have equal access to everything in the type. An instance does not contain a programming editor or code blocks; hence the code cannot be written inside an instance. All logic must be created in the type.

The allocated memory for creating a type solution (for example, a motor type solution that contains one motor type and 20 motor instances) is distributed mainly on the programming code inside the type. Therefore, the cost (allocated memory) for each new instance (motor) is very small, compared to the type itself. The instance only needs to allocate memory for variables, as the code is located and executed from the type. However, the number of instances are relevant for considering the total CPU memory.

It is easier to update the application while working with newer version of types, since the inherited mechanism takes care of changes that often concern hundreds of instances. A code change (for example, declaring additional connection parameters) can be done once for the type, and this change is inherited by all instances simultaneously.

Control Builder also contains a number of structured data types. For more information, refer to the *System 800xA Control AC 800M Planning (3BSE043732*)* manual. A type described in this sub-section is a function block type, a control module type, or a diagram type.

Define a Type in the Editor

Select the type from Project Explorer and open the corresponding Editor to declare the necessary parameters for the type.

The editor of a type contains several declaration panes that can be opened from the following tabs:

- Parameters
- Variables
- Function Blocks

- Control Modules (only for diagram type editors)
- Diagrams (only for diagram type editors)

Apart from the declaration panes, the editor contains:

- Programming editor for programming the code using IEC-61131 languages (see [Figure 11](#)).
- Graphical editor called CMD Editor (only for control module types, see [Figure 14](#)).
- Graphical programming editor for FD (Function Diagram) (only in diagram type editor, see [Figure 13](#))

Declaration Pane for Parameters

To open the declaration pane for parameters, double-click the type (to open the editor), and then select the **Parameters** tab.

[Figure 5](#) shows the editor for My_MotorType, with the declaration pane for parameters selected. These parameters can be used for connecting variables outside the instance.

	Name	Data Type	Attributes	Direction	FD Port	Initial Value	Description
1	out1	BoolIO					
2	FB1	BoolIO					
3							
4							

Parameters Variables External Variables Function Blocks

Figure 5. Declaration pane for creating connection parameters

Declaration Pane for Local Variables

To open the declaration pane for variables, double-click the type (to open the editor), then select the **Variables** tab. If the editor is already open, simply select the **Variables** tab.

[Figure 6](#) shows the declaration pane for creating local variables inside the type. The local variables can be used by the code inside the type.

	Name	Data Type	Attributes	Initial Value	Description
1	MotorStartTime	time	retain	10s	
2					
3					
4					

Parameters Variables External Variables Function Blocks

Row 1, Col 1

Figure 6. The declaration pane for creating local variables

Declaration Pane for External Variables

External variables are pointers to global variables. An instance can declare an external variable locally and then use this variable to access the value in a global variable located in the application. External variables and global variables are discussed in [External Variables](#) on page 96.

Declaration Pane for Communication Variables

Communication variables are declared in top level Diagram editor, Program editor, or top level Single Control Module editor. For details about communication variables, see [Communication Variables](#) on page 101.

[Figure 7](#) shows the declaration pane for communication variables in a Diagram editor.

	Name	Data Type	Attributes	Direction	Initial Value	ISP Value	Acknowledge Group	Interval Time	IP Address	Expected SIL	Unique Id	Description
1	a1	real	retain	in	1	0	1	fast	172.16.18.3	same	201	
2	b1	real	retain	in	1.1	2	auto	very fast	auto	SIL3	202	
3	c1	real	retain	out	1			slow			203	
4												

Variables Communication Variables Function Blocks Control Modules Diagrams

Figure 7. Declaration pane for communication variables

Declaration Pane for Function Blocks

To open the declaration pane for function blocks, double-click the type (to open the editor), and select the **Function Blocks** tab. If the editor is already open, simply select the **Function Blocks** tab.

Figure 8 shows the declaration pane for declaring function blocks inside the type.

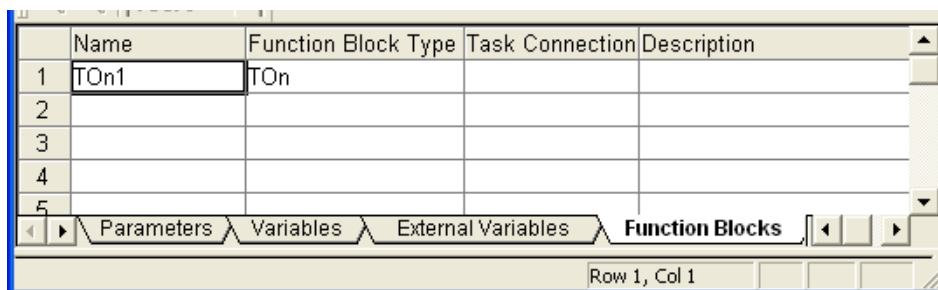


Figure 8. Declaration pane for creating function blocks inside a type

Enter the name of the function block in the Name column, and select the cell in the Function Block Type column. Press CTRL+J to open a context menu with all function block types available.



Connect all libraries with the required function blocks types to the application. Only then, the available function block types are listed in the context menu (CTRL+J)

Declaration Pane for Control Modules

To open the declaration pane for control modules, double-click the diagram or diagram type (to open the editor), and select the **Control Modules** tab. If the editor is already open, select the **Control Modules** tab.

Figure 9 shows the declaration pane for declaring control modules inside the diagram or diagram type.

	Name	Control Module Type	Task Connection	Description
1	Alarm_2322	AlarmCondM		
2				
3				
4				
5				

Variables Communication Variables Function Blocks **Control Modules**

Figure 9. Declaration pane for creating control modules inside a diagram type

Enter the name of the control module in the Name column, and select the cell in the Control Module Type column. Press CTRL+J to open a context menu with all control module types available.



Connect all libraries with the required control module types to the application. Only then, the available control module types are listed in the context menu (CTRL+J).

Declaration Pane for Diagrams

To open the declaration pane for diagrams, double-click the diagram type or diagram (to open the editor), and select the **Diagrams** tab. If the editor is already open, select the **Diagrams** tab.

Figure 10 shows the declaration pane for declaring diagrams inside the diagram type or diagram.

	Name	Diagram Type	Description
1	Valve101	Diagram_Area1	
2			
3			
4			
5			

Variables Communication Variables Function Blocks Control Modules **Diagrams**

Figure 10. Declaration pane for creating diagrams inside a diagram type

Enter the name of the diagram in the Name column, and select the cell in the Diagram Type column. Press CTRL+J to open a context menu with all diagram types available.



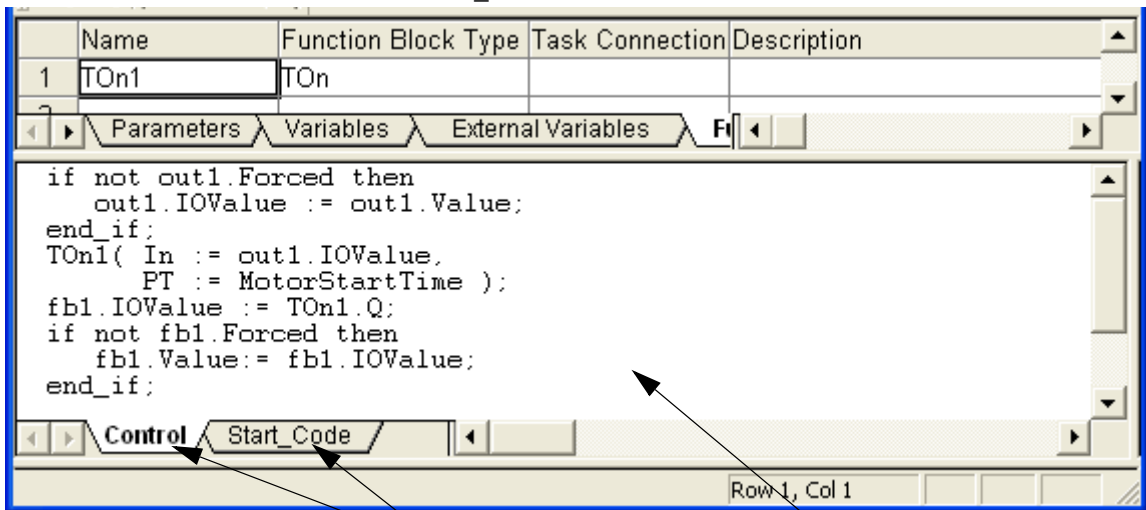
Connect all libraries with the required diagram types to the application. Only then, the available diagram types are listed in the context menu (CTRL+J).

Code Pane for Control Module Types and Function Block Types

The code pane in the editor for programs, control module types, function block types, and single control modules supports five programming languages that conforms to the IEC 61131-3 standard. The code pane is always active, and can be accessed irrespective of which tab is selected (parameters, variables, function blocks, etc.).

The code pane can be expanded using more code blocks up to 100 for structuring the code. These code blocks are then executed either in a predetermined order as decided by the compiler (control modules), or from left to right (function blocks).

Figure 11 shows a part of the code pane of a control module type editor. This code block uses Structured Text (ST) as the language. This editor contains two code blocks: Control and Start_Code.



Code blocks Code pane
 Figure 11. A code pane with two code blocks.

A brief description of code blocks in general and *Start_* code blocks:

- Code blocks are very useful for structuring the code. Dividing the programming code into a number of code blocks, improves the overall code structure and readability. Examples of code blocks are Control, Object Error, Operators, etc.



Code block names cannot contain certain characters. See Online help for information on characters that cannot be used in code block names.

- There is no limit to the number of code blocks that can be created in a type. Create only the required number of code blocks, since each code block affects the memory consumption and the execution time of the type.

- *Start_*

A code block with the prefix *Start_* is always executed first in an application and only once, at the application startup (after a warm and cold start, but not after a power failure).



The *Start_* code block is valid only for single control modules and control module types.

This code block must be used for initiating alarm strings, converting project constants to strings, etc.

However, there are some limitations while using the *Start_* code block:

- It is not suitable to place functions, function blocks, etc, in a *Start_* code block.
- It is valid only for the code blocks in control modules, and not for the code blocks in SFC (Sequential Function Chart).
- The `FirstScanAfterApplicationStart` function must not be used in the block.
- Function blocks for communication must not be used in the block.



If the application contains a very large chunk of code that has to be run in the first scan (for example, alarms in the *Start_* code block), the execution time can be so high that overrun occurs. This leads to the eventual shut-down of the controller.

Code Block Context Menu

Right-click a code block tab to access the code block context menu.

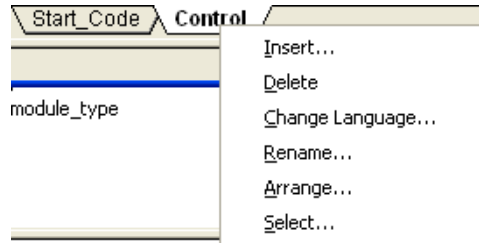


Figure 12. Code block context menu

Graphical Code Pane for Diagram Types

The code pane in the editor for diagram type supports the FD (Function Diagram) language. The FD code block in this editor allows mixing of functions, function blocks, control modules and other diagrams, through graphical connections, to create the logic. Variables and parameters can also be connected graphically. This represents a complete graphical overview of the whole logic.

The FD code block is always active, and can be accessed irrespective of which tab is selected in the declaration pane (parameters, variables, function blocks, control modules or diagrams).

The logic created in the default FD code block can be expanded using optional ST and SFC code blocks, which can be invoked in the FD code block or sorted separately.

To open this editor, right-click the diagram type, and select Editor. [Figure 13](#) shows an example logic created in the FD code block of diagram type editor.

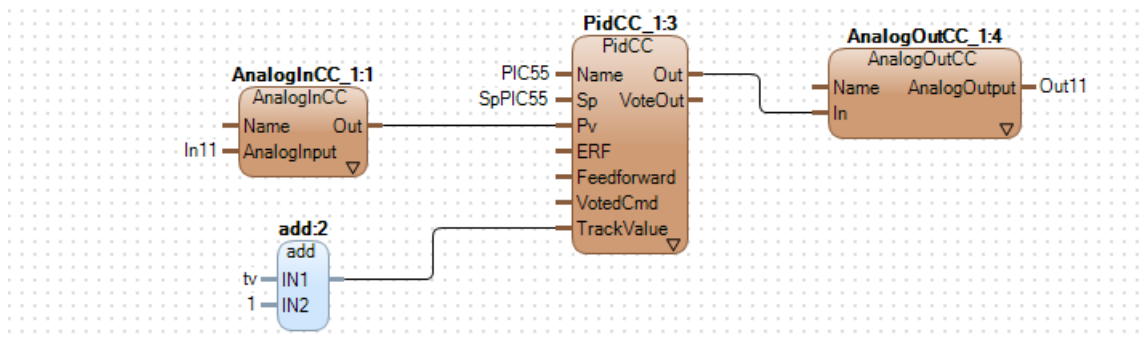



Figure 13. Example logic created in the graphical code pane of diagram type editor

Graphical Editor - CMD Editor

The graphical editor, Control Module Diagram Editor (CMD Editor) is also a combined editor for drawing and programming. The term ‘diagram’ refers to the graphical view of control modules and connections.

Use this editor to create and edit control modules, code, and graphics, and to connect variables and parameters.

To open the CMD Editor, right-click the control module type  and select CMD Editor. Figure 14 shows part of the graphical editor (CMD Editor).

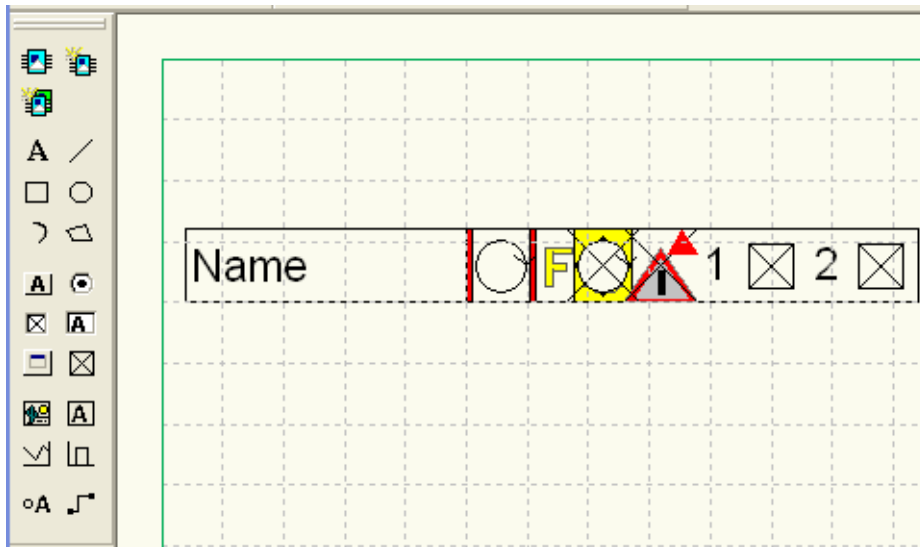


Figure 14. Graphical objects created in the CMD Editor.

The drawing functions in the CMD editor include basic auto shapes (lines, rectangles, etc.), ready-to-use interaction instances (option buttons, check boxes, etc.), and composite instances (trend graphs, string selectors, etc.). The graphical instances are dynamic, that is, with changing variable values, the points move, colors change, and numerical values are presented.

Control Module Types, Function Block Types, and Diagram Types

A type can be a control module type, a function block type or a diagram type. The usage of different types can be mixed. For example, a control module can be created inside a function block type (to add graphics), or a function block can be created inside a control module type (to execute a list of basic functions). In a diagram type, it is possible to create function blocks, control modules, and diagrams, to define the entire logic.

Table 4. Differences between types

Property	Differences		
	Function Block Type	Control Module Type	Diagram Type (supports mixing function blocks, control modules, and nested diagrams)
Container POU inside the application	Programs, Diagrams or Single Control Modules	Diagrams or Single Control Modules	Diagrams
Graphical connections between objects	Yes	Yes	Yes, including graphical connection to parameter/variable objects.
Code sorting	No	Yes	Yes. All code blocks, in control modules inside the diagram type, are sorted together with the code blocks outside the diagram type in the container Diagram.
Execution	Function blocks are executed from code. Therefore, a function block is executed once or several times per scan, or it is not executed at all.	Control modules are executed only once per scan.	The execution order shown in the diagram is followed. Codeblocks from control modules are sorted into the diagrams execution order according to control module sorting rules (writing into a variable is sorted before a read).

Additionally, the following properties apply to function block types:

- Parameter values on function block types are copied (except In_Out parameters and parameters having by_ref attribute, see [Function Block Execution](#) on page 71).
- Function block types are required when using extensible parameters (see [Extensible Parameters in Function Blocks](#) on page 127).

The choice between control module types, function block types, and diagram types depends on the context and environment. For guidelines about the use of control modules, function blocks and diagrams, refer to the *System 800xA Control AC 800M Planning (3BSE043732*)* manual.

Types in Applications

Creating a type in an application is the quickest and easiest way to get started. Before creating types in an application, no new libraries need to be created; use the available methods like connect libraries, create user defined data types, and select the object type to use (see [Decisions When Creating Types](#) on page 64). However, if a type is created directly in an application, it can only be used inside that application.

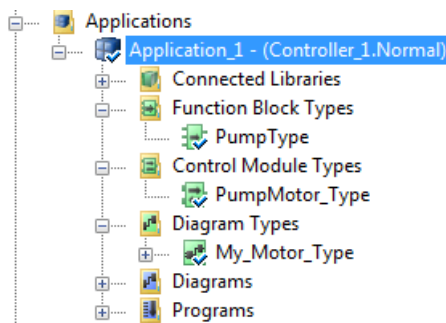


Figure 15. Types created under an application

To gain access to standard libraries (or user defined libraries), insert them into the control project (see [Library Management](#) on page 135), and connect them to the application. This allows the creation of instances in the application, from existing types in the connected libraries.

Types in User defined Library

The advantage of creating types inside a library, instead of creating them directly in an application, is the possibility to re-use them in other applications. If the types are created in a library, all the necessary functionality can be stored in this library. The library can then be connected to any application.

If a new library is created, user defined types can be created in that library (the 800xA System does not allow creation of types in a standard library).

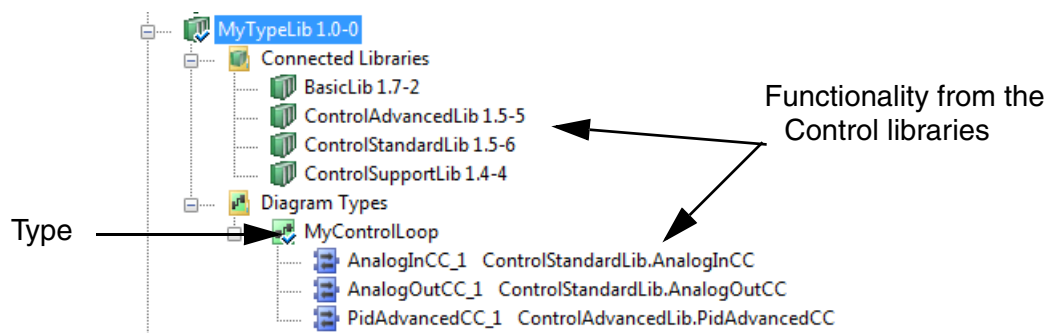


Figure 16. A Type (`MyControlLoop`) created in `MyTypeLib` library. This example shows a control loop created as a diagram type, while the components are ready-made instances from the standard libraries

Modify Complex Types

This subsection describes a use case where it is preferable to copy two types, instead of keeping a single and very large type in a library.

Refuse Incinerator Type - Problem

In this example, assume that a plant area has two identical refuse incinerators.

A type solution like this is manageable if a Refuse Incinerator type is created in a library with several underlying types. This type can then be re-used twice (as two objects), in two separate applications, by connecting the library to each application.

The following are the examples of underlying types inside the Refuse Incinerator type:

- A Feeder type containing 10 conveyors.
- A Combustion type.
- An Ash Handling type.
- A Flue Gas type.

After building the Refuse Incinerator type in the library, connect the library to both Application_1 and Application_2. This helps in creating an Incinerator1 instance in Application_1 and an Incinerator2 instance in Application_2.

If the Incinerator2 instance running in Application_2 suddenly needs an individual change (for example, 20 conveyors instead of 10 conveyors), edit the library and change the Feeder type inside the Refuse Incinerator type. But, changing anything inside the Refuse Incinerator type affects both incinerators due to the type and instance inherit mechanism.

By changing the Feeder type to include 20 conveyors, both the Incinerator instances are changed suddenly to contain 20 conveyors, which is not the intended use.

Refuse Incinerator Type - Solution

To avoid the problem, once the type is ready, consider the possible individual (instance) changes in the future. If an individual instance needs to be changed, copy the type on the highest type level (in this example, Refuse Incinerator Type1 and Refuse Incinerator Type2).

Create an Incinerator10 instance in Application_1, based on Refuse Incinerator Type1, and then create an Incinerator20 instance in Application_2, based on the new type copy, Refuse Incinerator Type2. This increases the memory consumption in the controller, but allows individual changes. For example, the number of conveyors in the feeder for one of the applications can be changed, without affecting the other.

Diagram and Diagram Types

Diagrams are created under an application, and diagram types (which can be reused as instances in a diagram) are created under a library or under the same application as the diagram.

The FD code blocks in diagrams and diagram types allow mixing of functions, function blocks, control modules, and other diagrams, and graphically connect them to achieve a particular logic.

[Figure 17](#) shows the workflow for using diagrams and diagram types.

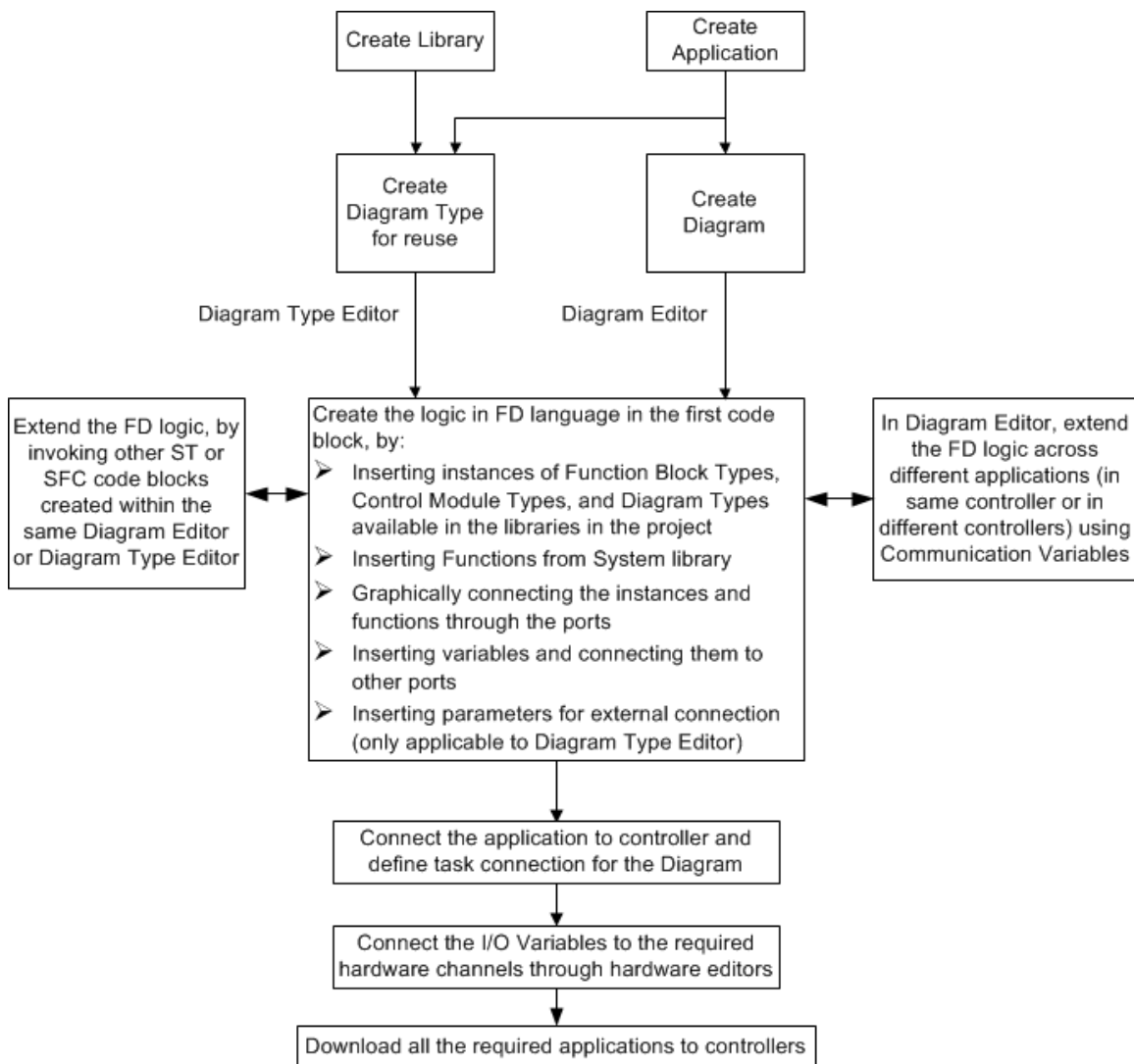


Figure 17. Workflow for using Diagrams and Diagram Types

Figure 18 shows the editor for a diagram POU under an application.

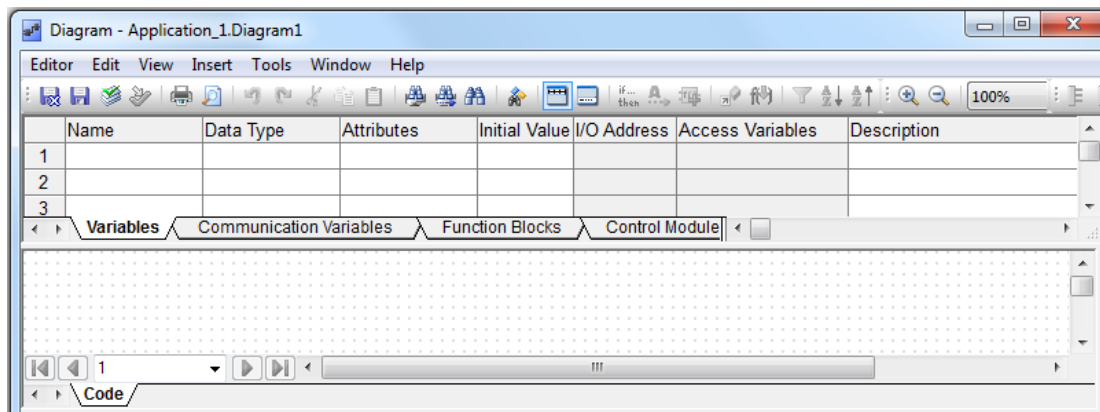


Figure 18. Editor for Diagram POU

The diagram editor consists of declaration pane and code pane. The code pane contains a grid area where you add the objects and create graphical connections.

The editor for diagram type also looks similar. The editor for a top level diagram (under the application) differs from a Program editor in following ways:

- The first code tab is always a FD language tab.
- Additional code block tabs can be created, but only for ST and SFC.
- Two additional tabs – **Control Modules** and **Diagrams** – are available in the declaration pane.

The editor for a diagram type (under the application or library) differs from a diagram editor in following ways:

- There is a **Parameter** tab instead of a Communication Variables tab in the declaration pane.
- There are no I/O Address or Access Variables columns in the **Variables** tab.



The diagram editor or diagram type editor has one mandatory FD code block. Only one FD code block is allowed within this POU. It is allowed to have several optional ST and SFC code blocks in the diagram editor or diagram type editor.

Figure 19 shows the editor for a diagram after it is edited (to create an example logic).

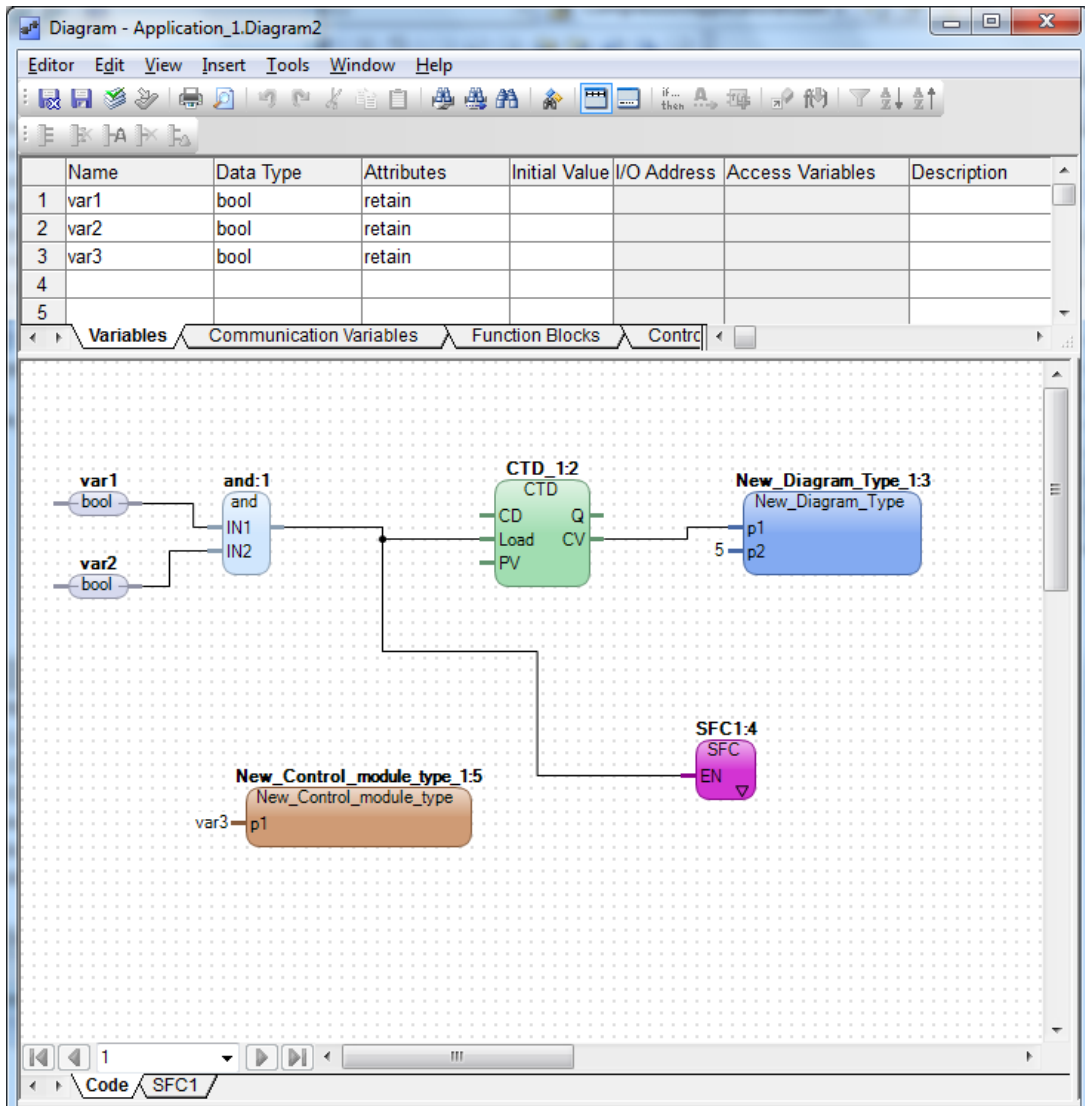


Figure 19. Editor for diagram POU with an example logic created

Characteristics of Diagrams and Diagram Types

The following are the main characteristics of diagrams and diagram types:

- A diagram or diagram type can contain other diagrams (which are the instances of other diagram types), in addition to functions, function blocks, and control modules.
- The objects inserted in the diagram editor or diagram type editor can be connected graphically through ports, to create the logic.
- The port of an object inserted in the diagram editor or diagram type editor can be connected to variable by entering the name of the variable or by graphically connecting the port to the variable object.
- The FD code block in the editor also supports Split blocks and Join blocks to work with objects having structured data types:
 - A Split block splits the structured data type variable, connected as its input, to its components that are displayed as output connection ports. These output connection ports can be connected to variables based on the data types.
 - A Join block displays input connection ports, which are the components of the structured data type, and these can be connected to variables. The output connection port is connected to the structured data type variable.



Split and Join blocks does a copy of the variable in run-time. To avoid a copy, create a variable and make the components visible, or do component connections directly.



Split and Join blocks must not be used if the structured data type has components with *reverse* attribute.

For the Split block, if *reverse* components are used, a change in the extracted (split) data of a *reverse* component does not result in a change of this component in the structured variable. For the Join block, if *reverse* components are used, it is not possible to write to an *out* variable using Join, even if the attribute is *reverse*.

- The FD code pane supports creation of additional pages for adding more objects and connects the objects graphically across pages. This helps to extend the logic from the default page.

- The logic created in the FD code pane can be extended through connections to other ST and SFC code blocks in the editor.
- The FD logic can be extended across different applications by using communication variables declared in the diagram editor. These variables support cyclic communication between the top level diagrams, top level single control modules and programs in different applications.
- The FD code block in a diagram is the only code block in Control Builder that supports a lower SIL input signal to be used in a higher SIL application. This is done using graphically connected communication variables in the editor.

Decisions When Creating Types

This subsection describes the decisions to be made about the types before programming the code, and declaring parameters and variables. Many functions and type solutions have been developed already, and the Control Builder helps to set up and access these options before programming. Read more about design analysis in the *System 800xA Control AC 800M Planning (3BSE043732*)*.

The following decisions must be made before creating the types:

- Whether there is a need to create instances in user defined type(s).
These types are based on other types located in external libraries. In that case, those external libraries must be connected to the library or application.
- Whether there is a need to create self-defined structured data types for passing parameters through several layers of instances.
The data types are automatically connected to the library or application. Structured data types are often useful in more complex type solutions, with a deep hierarchical structure.
- Whether a function block type or a control module type or a diagram type should be used.
 - If the code is programmed in the Program POU¹ only, select function block types.
 - If a graphical editor is preferred for programming the code, and automatic code sorting is also preferred, select control module types.

1. See [Program Organization Units, POU](#) on page 23.

- If a graphical editor is preferred for programming the code, and automatic code sorting as well as mixing of functions, function blocks, control modules and nested diagrams are also preferred, select diagram types. This helps you to overcome the limitations when using a single type.



For information on how to access these methods, refer to the Control Builder online help. Select one of the folders in Project Explorer and press F1.

Create and Connect Instances

An instance is a function block, control module or diagram, based on a type.

Each time a new instance is created, the Control Builder prompts for a type. The type can be located in an inserted library (inserted into the control project), user defined library, or directly in an application. In any case, a type and its location must always be selected.

Once the type is selected, connect the connection parameters.

[Figure 20](#) shows the creation of an instance (Pump10) based on My_MotorType, which is a diagram type located in the application. The instance needs the location (Application_1) and the type (My_MotorType).

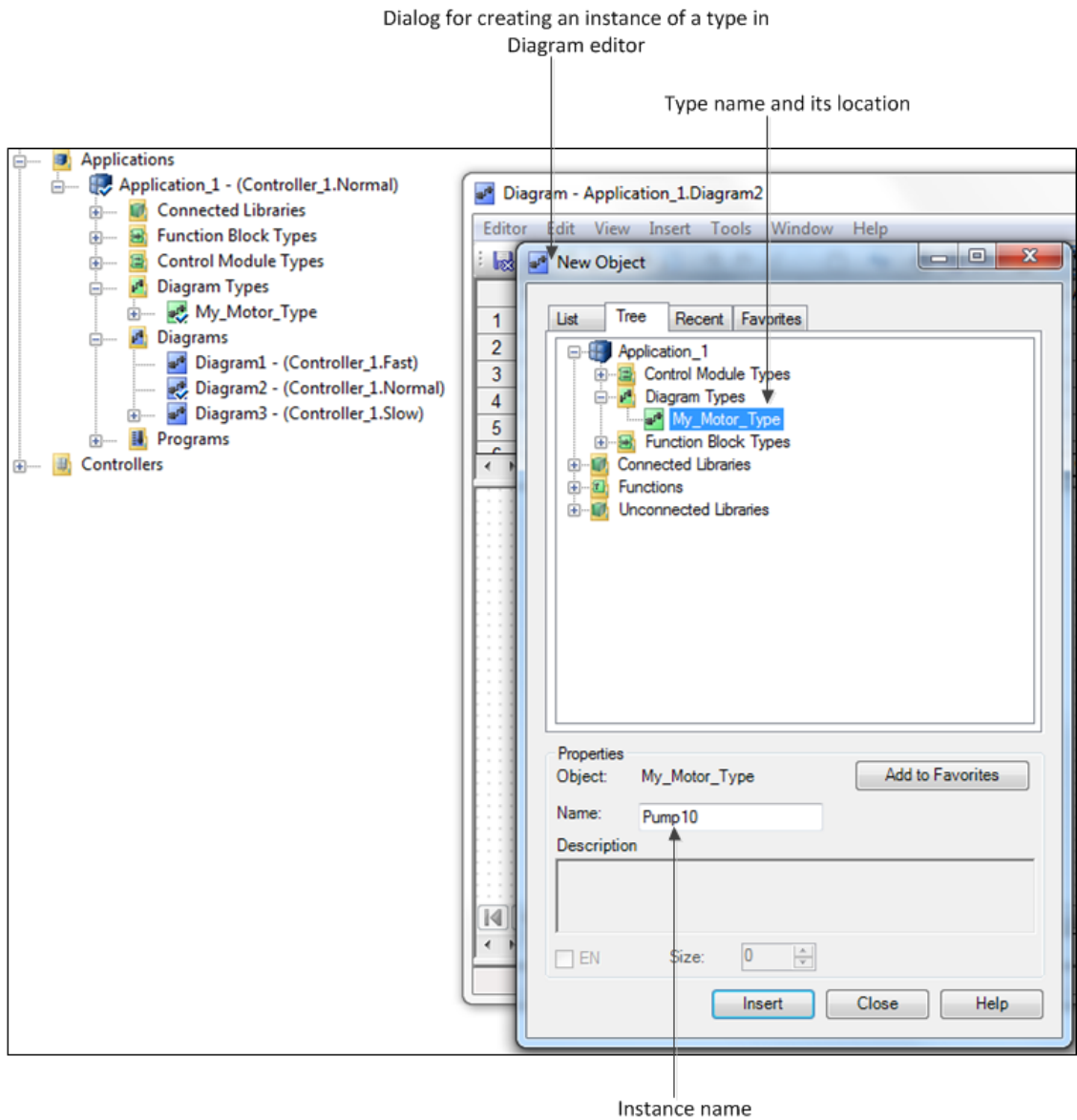


Figure 20. Creating instance of a diagram type in Diagram editor

Connections Using Parameters

The instances can be connected either through graphics or through text, using the parameters in their respective types.

Control Modules

For control modules (instances of control module types), graphical connections are done directly in the Control Module Diagram (CMD) editor and textual connections are implemented in the Connection editor.

Diagrams

For diagrams (which support instances of diagram types, function block types, and control module types), both graphical connections and textual connections of instances are done directly in the FD code block of the diagram editor.

Graphical Connections in CMD Editor

Graphical nodes and graphical connections in CMD editor connects the control modules effectively.

The control module parameters, which can be graphically connected, contains `NODE` in the beginning of the parameter description. This is the standard for all control modules located inside the standard libraries.

Nodes for graphical connections can also be created for self-designed control modules. Graphical connections are suitable for obtaining a comprehensive view of main flows, for example, in a PID controller or for group start of several motors. [Figure 21](#) shows three graphical connections for group starting motors. The modules are connected using the Graphical Connection function (located in the CMD Editor).

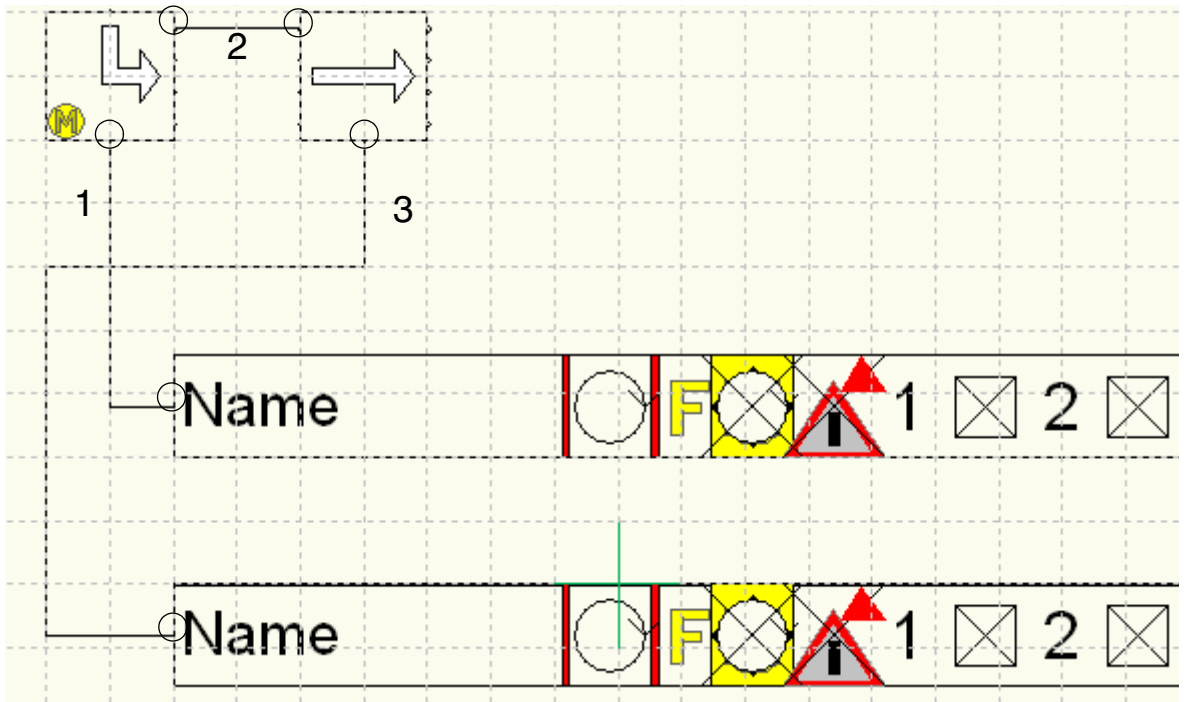


Figure 21. Two motor instances that have been graphically connected with a Start and Next instance located in the Group Start library. The circles symbolize the connection nodes

Textual Connection for Control Modules

To open the Connections editor via the Connections entry, right-click the control module (instance of control module type) and select **Connections**.

Parameters can be connected to the actual variables presented in the Connections editor. Textual connection is the only way to connect parameters when the control module is subordinate to a function block, since there are no surrounding graphics.



It is not possible to connect the same parameter both graphically and textually.

Connect an instance of Control Module in Connections Editor

The Connections editor is a parameter/variable interface between the instance and its closest surrounding. The Connections editor displays the parameters that are declared in the type, with reference to the control module instance, and connects the surrounding parameters/variables to the instance.

If a control module instance is created in an application (see Figure 22), then the application can be seen as the closest surrounding, and the variables in the application must be connected to the instance.

If a control module instance is created in a type (located in a library), then the type can be seen as the closest entity, and parameters/variables in the type must be connected to the instance.



To connect the parameters to instances located several hierarchical layers away (not the closest), use structured data types that simplifies the connections (instead of passing corresponding parameters). For more information on structured data types, refer to the *System 800xA Control AC 800M Planning (3BSE043732*)*

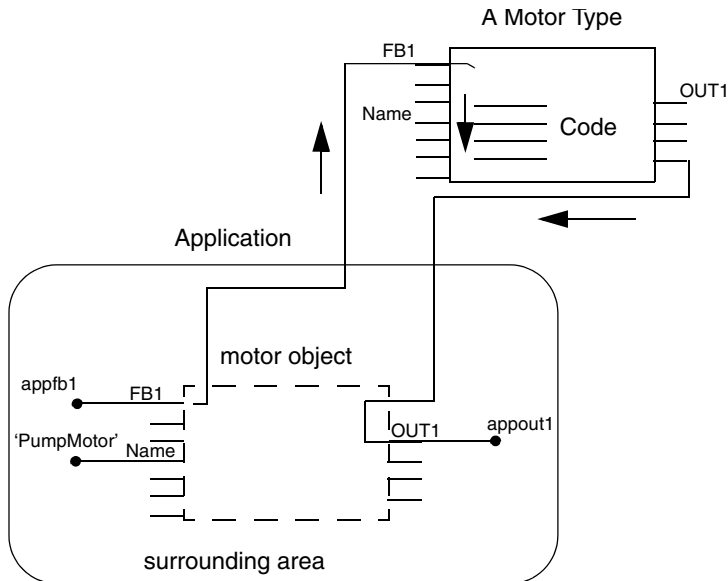


Figure 22. A control module instance connected to variables in an application. The application is the 'surrounding area' with the variables appfb1, Name (initial value 'PumpMotor') and appout1 connected to the instance.

In [Figure 22](#), the connection parameters for the motor instance connect the parameters (*FBI*, *Name* and *OUT1*) to the variables (*appfb1*, *appout1*, *Name*; *Name* has the initial value PumpMotor) that have been declared in the application.

Function Block Execution

There are three types of function block parameters: In, Out, and In_out.

The input and output parameters are passed by value, which means that the function block creates copies of each variable value, before and after the function block is executed. The *In_Out* parameters are passed by reference, which means only a reference to the actual variable outside the function block is passed to and from the function block.

Input parameters create a copy of each variable before the function block executes, and the output parameters create a new copy after the function block is executed and pass the new values to the surrounding variables outside the function block.

For complex data types and strings, a reference to the data instance can be passed in the function block call. This is achieved by setting the attribute of the parameter to *by_ref*.

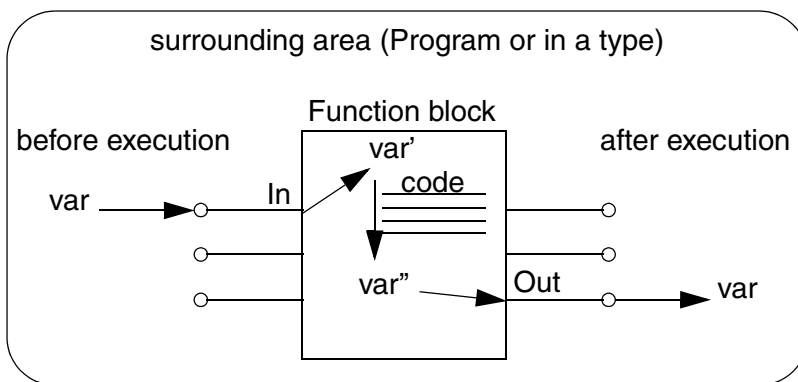


Figure 23. In and Out parameters for a function block. This example illustrates how In and Out parameters copies the variable (var).

Using *by_ref* on parameters enhances the performance. It takes a lot of execution time to copy parameters in each scan.

There are some limitations when using *by_ref*:

- It is not possible to connect expressions or literals to a reference parameter.
- If a reference parameter is not connected in one invocation, it cannot be connected in other invocation (if the instance has multiple invocations).
- It is not possible to read or write the parameter from outside the function block (except in the invocation). The example expressions like `fb.par_in := 2;` or `k := fb.par_out;` are not allowed for reference parameters.

By using *by_ref*, it is still possible to use *init* values, in which case the *init* value is the default value. If the parameter is not connected, the default value is used.

The code generated for connecting *by_ref* parameter is identical to an *in_out* parameter; but they differ in what is allowed inside the function block.

For example, it is not allowed to write onto an *in* parameter regardless of whether it is a reference or value parameter. The ownership analysis detects that a variable is read only if an *in* parameter by reference is used instead of *in_out*. It is therefore preferable to use `direction=in` and `attribute=by_ref` (instead of *in_out*), if the parameter is actually an *in* parameter.

The *In_Out* parameters are passed by reference, and only a reference to the actual variable outside the function block is passed to and from the function block. The local representation of the parameter does not exist inside the function block. Performing operations on an *In_Out* parameter inside a function block means performing operations directly on the actual variable connected to the function block. See also [Connecting Variables to I/O Channels](#) on page 122.

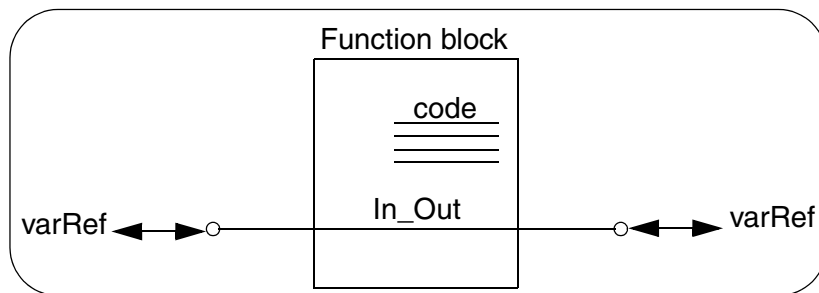


Figure 24. *In_Out* parameter for a function block. This example illustrates how the *In_Out* parameter points as reference to the value in the variable *varRef*.



A structured data type having components with *reverse* attribute must not be used for communication between function blocks. The components with *reverse* attribute does not work as intended when used with function blocks.

Control Module Execution

Control modules provide data flow-driven execution, which makes the code design much easier for solutions where several types and formal instances are needed. All control modules communicate with each other, and can therefore determine when each individual instance can send and receive information. A data flow-driven design prevents possible mistakes, when trying to foresee the correct execution order, since the compiler rearrange or sort all the code behind the scenes. This is called code sorting.

Direction for Control Modules

In control module types, a parameter can have any of the following direction:

- In
- Out
- In_out
- Unspecified.

These control module parameters follow different access rules from the code inside the control module and offer limitations to the methods used to connect them.

All of them are passed by reference, which means only a reference to the actual variable outside the control module is passed to and from the Control module.

The rules governing their functioning are as follows:

- Input parameters are read only.
- Out, In_Out and Unspecified are read and edit.
- Control modules on the same level can connect only In to Out.
- A sub control module inside could only connect its In parameters to In parameters in the surrounding control module and so on.
- In_out must be connected to a variable (on any level). This is not the case if the control module is used in FD code block in a diagram.

- If the control module is used in FD code block in a diagram, the In_out parameters can be connected to each other in FD code block. It is also possible to connect one Out parameter to an In_Out parameter, and an In_Out to an In parameter.
- One Out can be connected to Several In parameters. But, it is not possible to have multiple data connections from the same source on control module parameters of structured data types that have *reverse* components. See [Table 9](#).
- A control module is allowed to write to an output parameter. An exception is the case of output parameters that are of structured data type containing components with the *reverse* attribute. It is an error if a control module writes to a *reverse* component of an output parameter.
- A control module is not allowed to write to an input parameter. An exception is that it is allowed for a control module to write to a *reverse* component of an input parameter.

These rules apply to connecting parameters to communication variables as well. Communication Variable In should be connected to In parameters and the corresponding for Out. The compiler (and check) warns if rules are broken.

Unspecified parameters can be used without limitations for compatibility reasons.



For more information on Code Sorting, see the *System 800xA Control AC 800M Planning (3BSE043732*)*.

Diagram Execution

The execution of the content in a diagram or diagram type is mainly configured using the Data Flow Order of different invocations within the FD (Function Diagram) code block. The Data Flow Order is a number that specifies the intended order of execution. In the FD code block, the Data Flow Order is given to invoke functions, function blocks, diagram instances, control modules, code blocks, split blocks and join blocks.

Control modules in a diagram are sorted based on access of variables to enable both forward and backward calculations and data flows to be executed in the same task scan. Therefore code blocks in invoked control modules will not always be executed in the order specified by the Data Flow Order.

The Structured Text and SFC code blocks can be defined without invoking them from the FD code block. These code blocks are then sorted together with code blocks of invoked control modules.

Direction for Diagram Types

In diagram types, a parameter can have any of the following direction:

- In
- Out
- In_out

The diagram type parameters are shown as ports when the type is instantiated in the diagram editor.

All of them are passed by reference, which means only a reference to the actual variable outside the diagram type is passed to and from the diagram type.

The rules governing their functioning are as follows:

- Each port has an attribute that determines if it is visible in the diagram or not.
- Input ports are shown on the left side.
- Output ports are shown on the right side.
- In_out ports are normally shown on both sides with a connecting line through the block. They can also be shown on the left side only, depending on the FD Port property on the corresponding parameter declaration.
- One Out can be connected to Several In parameters. But, it is not possible to have multiple data connections from the same source on diagram type parameters of structured data types that have *reverse* components. See [Table 9](#).
- A diagram type is allowed to write to an output parameter. An exception is the case of output parameters that are of structured data type containing components with the *reverse* attribute. It is an error if a diagram type writes to a *reverse* component of an output parameter.
- A diagram type is not allowed to write to an input parameter. An exception is that it is allowed for a diagram type to write to a *reverse* component of an input parameter.

These rules apply to connecting the parameters to communication variables as well. Communication Variable In should be connected to In parameters and the corresponding for Out. The compiler (and check) warns if rules are broken.

Single Control Modules

A special kind of control module type, the single control module, provides a way of grouping graphical instances, variables, parameters, and control modules into a single unit.

Compared to the previous discussions about types and instances, a single control module can be considered as a hybrid of them both (see [Figure 25](#)). First of all, create a single control module as an instance under the control module folder (not the control module type folder) in an application.

Once a single control module is created, it starts acting as both a type and an instance. It contains code, editors for declaring parameters, function blocks, instance information, etc. just like a regular type or instance. A single control module can never be reusable as a type that can be used to create many instances. However, it can be copied to a new single control module, and then be modified.

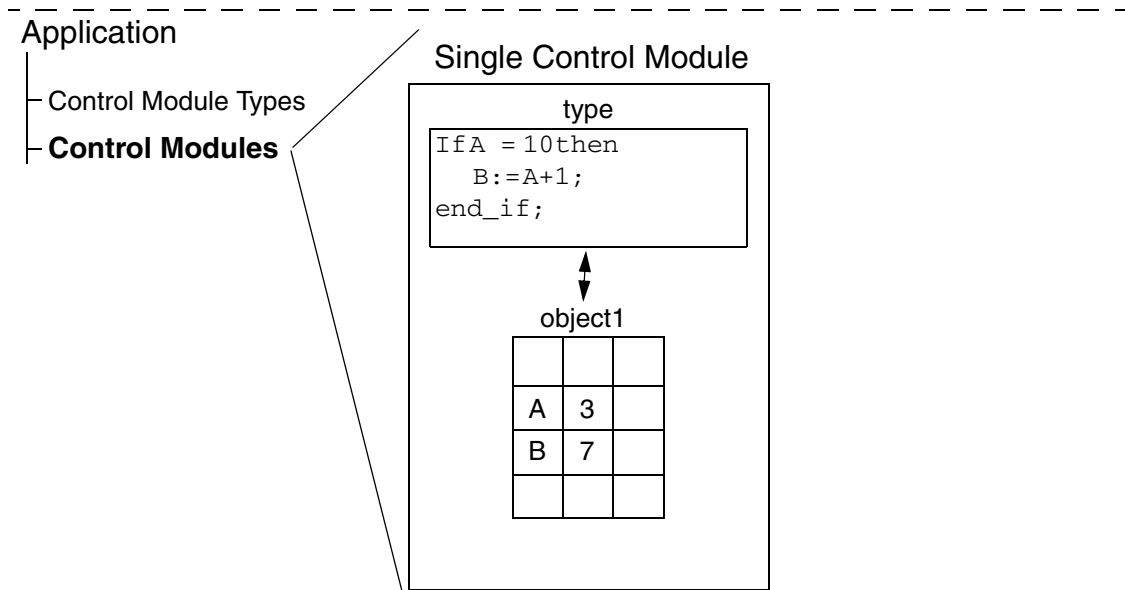


Figure 25. A single control module. This module is not reusable, hence intended to be used only once for grouping instances into a single unit.

Single control modules can be used as a framework and attach control module instances inside, like an application does with instances. Figure 26 illustrates this, where three single control modules (Transport, Heating, and Crushing) form the framework for the control modules (Motor_1, etc.).

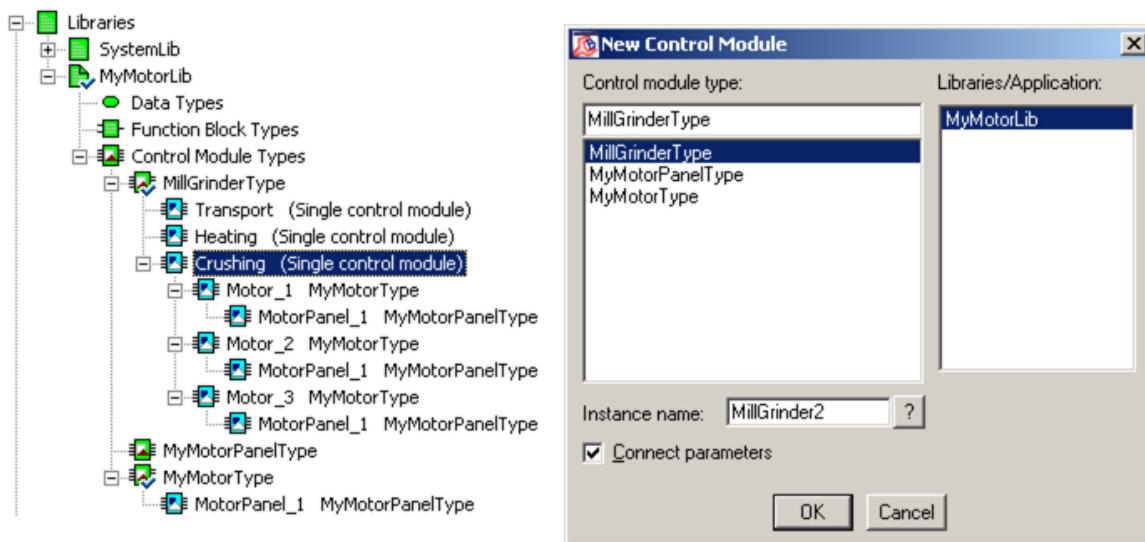


Figure 26. Single control modules form the framework for the control modules

FD Port

The FD Port column appears in the editor for function block types, control module types, and diagram types. This column is only significant for the types that are instantiated in a Function Diagram (FD) code block.

The normal choice is *Yes* or *No*. The value specifies if the parameter shall be visible when the function block type, control module type, or diagram type is instantiated in an FD code block. The default value is *Yes*.

There are extra choices (*Left* or *Right*) for control module parameters with direction *unspecified* and parameters with direction *in_out*. These choices are related to the placement of the parameter port in the FD code block.

Unspecified parameters are placed on the left side by default, and *in_out* parameters are placed on both sides by default.

There are some types with structured parameters that are mostly output, but also contain some input components. Such a parameter must be either an Unspecified parameter (control module types only) or an In_Out parameter.

The following list of alternatives are available for parameters of direction *unspecified*:

- *No* - Not visible as a port. The parameter will be placed on the left side of the object if the user decides to show it later on.
- *No Left* - Not visible as a port. The parameter will be placed on the left side of the object if the user decides to show it later on.
- *No Right* - Not visible as a port. The parameter will be placed on the right side of the object if the user decides to show it later on.
- *Yes* - Visible as a port on the left side of the object.
- *Yes Left* - Visible as a port on the left side of the object.
- *Yes Right* - Visible as a port on the right side of the object.

The following list of alternatives are available for parameters of direction *in_out*:

- *No* - Not visible as a port. The parameter will be placed on both sides of the object if the user decides to show it later on.
- *No Left* - Not visible as a port. The parameter will be placed on the left side of the object if the user decides to show it later on.
- *No Right* - Not visible as a port. The parameter will be placed on the right side of the object if the user decides to show it later on. This is only available for parameters of function block types.
- *Yes* - Visible as a port on both sides of the object.
- *Yes Left* - Visible as a port on the left side of the object.
- *Yes Right* - Visible as a port on the right side of the object. This is only available for parameters of function block types.



It is **not** recommended to use a function block type parameter of direction *in_out* with its FD Port property set to *Yes Right*. In this case, it is only possible to connect this type of parameter to a variable, parameter, or communication variable.
Therefore, it is recommended to use parameter with direction *out* and attribute *by_ref*, instead of *in_out* with *Yes Right* option, so that this parameter can be connected to input port of another object.

Aspect instances

Aspect instance is an attribute that decides whether the instance will be visible in Plant Explorer, or not.

The instances not interacting with other instances in Plant Explorer should have the aspect instance attribute set to False for not loading the Aspect Server performance.



Function blocks and control modules created from Plant Explorer will be aspect instances by default, regardless of the type is an alarm owner or not.

Set Aspect instance Attribute

To set the aspect instance attribute:

1. In Project Explorer, right-click the function block or control module and select **Properties > Aspect Object**. Use the check box to set the attribute (checked=True, unchecked=False).

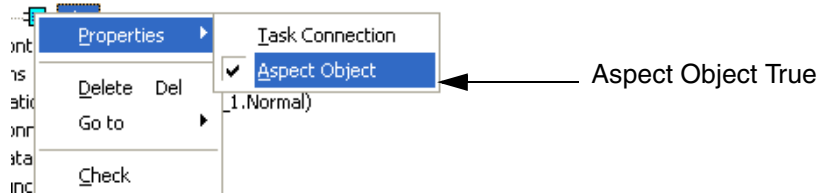


Figure 27. Aspect Object



Every time the **Aspect Object** menu item is selected, the aspect instance property is toggled on/off (true/false).

Suppress Aspect Object (Set Attribute to False)

If the attribute is set to false, the instance will not be visible in Plant Explorer and no live data can be fetched from the instance. If the instance has the aspect object attribute set to false, it cannot be accessed from Plant Explorer.



If the aspect object attribute is set to false, added aspects will be deleted without warning. Also, ensure that all editors are closed before changing this attribute in Project Explorer, otherwise there is a risk that aspect object settings are overwritten when the editor is closed.

Aspect Object (Set Attribute to True)

If the attribute is true, the instance will be visible in Plant Explorer (provided that the surrounding type is not hidden or protected). See also [Hide and Protect Control Module Types](#), [Function Block Types](#), [Diagram Types](#), and [Data Types](#) on page 158.

Set Instantiate as Aspect Object Attribute for a Type

- In Project Explorer, right-click the type and select **Properties > Instantiate as Aspect Object**. Use the check box to set the attribute (checked=True, unchecked=False).

Variables and Parameters

Variables and parameters are the carriers of data throughout the system. This section describes how to use parameters and variables in the best way possible:

- [Variable and Parameter Concept](#) on page 83 gives an overview of variables and parameters and how they are used.
- [Variables](#) on page 84 gives an overview of the different variable types.
- [Variable Entry](#) on page 85 describes how to declare variables.
- [Specific Initial Values](#) on page 94 describes how to use specific initial values.
- [External Variables](#) on page 96 describes how to define external variables.
- [Access Variables](#) on page 97 describes how to define and use access variables.

- [Communication between Applications Using Access Variables](#) on page 99 and [Communication in an Application Using Global Variables](#) on page 100 describe how communicate between applications.
- [Communication Variables](#) on page 101 describes how to define communication variables.
- [Control the Execution of Individual Objects](#) on page 113 describes how to use variables and parameters to control the execution of objects.
- [Link Variables in Diagrams](#) on page 116 describes the use of link variables for graphical connections in diagram editor.
- [Project Constants](#) on page 116 describes the use of project constants and how to update them.
- [I/O Addressing Guidelines](#) on page 121 describes the rules for addressing I/O channels.
- [Connecting Variables to I/O Channels](#) on page 122 describes how to connect I/O variables to I/O channels.
- [Extensible Parameters in Function Blocks](#) on page 127 describes extensible parameters (these can only be used in function blocks).
- [Keywords for Parameter Descriptions](#) on page 128 describes keywords used in description in editors to identify the function of a parameter.
- [Property Permissions](#) on page 133 describes how to set permission for variables and objects.
- [Property Attribute Override](#) on page 134

Variable and Parameter Concept

Variables

Variables are used in Control Builder to store and compute values.

Variables are the carriers of values at object level, application level, and network level:

- Local variables – These are mainly used inside objects as carriers of local values. They belong to the code and can only be accessed within the same function block, control module, diagram or program.
- Global variables – These are declared in the application and holds values that can be accessed by any object (function block, control module, or program) in the application. However, to reach a global variable, each object that intends to use a global variable must have declared a corresponding External variable, see also [External Variables](#) on page 96).
- Access variables and Communication variables are used as carriers for communication between several applications and controllers in a network:
 - Access variables allow data exchange between controllers, that is, access variables can be accessed by other controllers. See [Communication between Applications Using Access Variables](#) on page 99.
 - Communication variables are used for cyclic communication between top level diagrams, programs, and top level single control modules. Communication variables support both inter application communication and inter controller communication in a system network. For more information, see [Communication Variables](#) on page 101.

All variables are defined by their names and data types. The data type (dint, bool, real, string, and so on) defines the characteristics of the variable.

Parameters

Parameters cannot store any values, but the variables are assigned to parameters of function blocks, control modules, diagrams and functions. Variables store the value of the corresponding (connection) parameters.

Use parameters for connecting objects and to point to variable values that need to be read into code blocks and written from code blocks.



When function blocks read from a variable and write to a variable, they use input and output parameters that temporarily copy the variable value, before and after execution. In this case, one may claim that parameters can temporarily hold a value. See [Function Block Execution](#) on page 71 for more details.

Variables

[Table 5](#) lists available variables in Control Builder.

Table 5. Variable types in Control Builder.

Variable type	Scope	Where to declare
Local variable	Object level. Can only be accessed within the function block, control module, diagram, or program in which it is declared.	Application editor (for passing parameters between control modules) or, Programs editor (for access in the program). Function block editor (for access inside the function block). Control module editor (for access inside the control module). Diagram editor (for access inside the diagrams)
Global variable	Application level. Can be accessed from anywhere in the code within an application, except from diagrams. An object that intends to use a global variable must declare an external variable locally that will point at the corresponding global variable.	In the application editor. See also Communication in an Application Using Global Variables on page 100.

Table 5. Variable types in Control Builder. (Continued)

Variable type	Scope	Where to declare
Access variable	Network level. Variable that can be accessed by remote systems for communication between controllers. See also Access Variables on page 97 and Communication between Applications Using Access Variables on page 99.	Access Variable editor of a controller.
Communication Variable	Network Level. Variable that can be accessed by remote systems for communication between applications and controllers. See Communication Variables on page 101	Editor for top level Diagram, Program, or top level Single Control Module.

Variable Entry

Control Builder helps the user to declare variables in applications, programs, function block types and control module types. This section covers the entries: Name, Data Type, Attributes, Initial Value and Description.

Name

It is recommended that variables are given simple and explanatory names, and that they begin with a capital letter. Names consisting of more than one word should have capital letters at the beginning of each new word. Examples of recommended variable names are DoorsOpen, PhotoCell.

Certain names, however, are reserved by the system and cannot be used for other purposes, for example *true*. An error message appears if such a word is used. For naming guidelines and information on relevant tools, refer to the *System 800xA Control AC 800M Planning (3BSE043732*)*.

Data Types

A data type defines the characteristics of a variable type. There are both simple and structured data types in Control Builder. A variable of simple data type contains a single value, while a structured data type contains a number of components of simple or structured data types.

Table 6 presents the most common simple data types and the initial value when the variable is declared.

Table 6. Simple data types

Data type	Description	Bytes allocated by variable	Initial value (default)
bool	Boolean	4	False, 0
dint	Double integer	4	0
int	Integer	4	0
uint	Unsigned integer	4	0
string ⁽¹⁾	Character string ⁽²⁾	10 bytes + string length [n]	“
word	Bit string	4	0
dword	Bit string	4	0
time	Duration	8	T#0s
date_and_time ⁽¹⁾	Date and time of day	8	1979-12-31-00:00:00
real ⁽³⁾	Real number	4	0.0

- (1) It is allowed to use variables of string and date_and_time also in SIL3 applications; however, the result must never influence the safety function of a SIL certified application. The variables cannot be send via safe peer to peer MMS, as SIL data.
- (2) String length is 40 characters by default, but can be changed by entering string[n] as the data type, where n is the string length. The number of bytes allocated for string[40] will be (40 +10) 50. The maximum string length is 140.
- (3) Implemented according to IEEE 754, single precision floating point. See Real value in AC 800M on page 129.



Comparison of variables of unsigned data types (uint, word, and dword) will not work properly if the most significant bit is set. Internally, they are handled as signed, where the most significant bit is used as the sign. This means that a word variable with a value above 32767 will be considered to be smaller than a word variable with a value below 32768.

When declaring variables or parameters of the data type string, always define the required length within square brackets (for example, string[20]), to minimize allocated memory. If the string length is not defined, then Control Builder automatically allocates memory for a 40 character string length.



Use variables of data type string with care. Strings occupy a great deal of memory, and require much execution time to be copied or concatenated.

A structured data type contains a number of components of simple or structured data type. For bidirectional communication using structured data types, a `reverse` attribute must be set to indicate which components communicate in the opposite direction (see also [Bidirectional Communication Variable](#) on page 107).

There are a number of predefined data types in Control Builder (for example BoolIO and RealIO) that are structured data types. User-defined structured data types can also be created, see [Decisions When Creating Types](#) on page 64.



The word “default” can be used as an initial value for a parameter in a control module type or diagram type. This works for both simple and structured data types. For a structured data type, the initial value “default” gives the default value of the data types for all components.

This is useful when creating types; for input parameters of a structured data type that do not have to be connected, and for output data types that do not have to be connected.



More information is given in Control Builder online help. Search the index for “structured data type”.

Attributes

Attributes are used to define how variable values should be handled at certain events, such as after cold restart, warm restart, etc. Variables that are supposed to hold values over several downloads must for example, have a retain attribute in order to keep their values after a warm start. Any of the attributes in [Table 7](#), can be given to a variable. For parameter attributes see [Table 8](#).

Table 7. Variable attributes

Name	Description
no attribute	The variable value is not maintained after a restart, or a download of changes. Instead, it is set to the initial variable value. If the variable has no initial value assigned, it will be assigned the default data type value, see Table 6 on page 86 .
retain	The variable value is maintained after a warm restart, but not after a cold restart. Control Builder sets retain on all variables by default. To override this, the attribute field must be left empty in declaration pane.
coldretain	The variable value is saved in the aspect directory, and retained after warm or cold restart. ⁽¹⁾ Coldretain overrides the retain attributes in a structured data type.
constant	The user cannot change the value online once assigned. This attribute overrides the coldretain and retain attributes in a structured data type.
hidden	The variable will be hidden for an OPC client connected to an OPC server for AC 800M. This attribute is used for variable values not necessary to a supervisory system.

Table 7. Variable attributes (Continued)

Name	Description
nosort	This attribute suppresses the code sorting feature for control module types. It is advisable not to use the nosort attribute if the user do not know the data flow characteristics in detail.
state	<p>This attribute will let the variable retain its old value between two scans for control module types. The old and new value can be read by adding <code>:old</code> and <code>:new</code> to the variable name. When a variable of state is declared will there automatically be an invisible code block created there the state variables handles in following way: Variable:old := Variable:new. These code blocks always execute first in an 1131 task. This means that the state variables, <code>:new</code> and <code>:old</code> always is equal when the first user defined code block starts to execute. Only the first change during code execution could be detected.</p> <p>State variables are for breaking genuine sorting loops. Not genuine sorting loops shall been handled by divide up the code block in two parts.</p> <p>The <code>:new</code> variable is default when writing to, or reading from state variables. Therefore is it recommended to write both <code>:new</code> and <code>:old</code> in the 1131 code since it will be much easier to understand which variable that concerns.</p> <p>It is not possible to detect changes on state variables between executions as they are equal when the code execution starts.</p>

- (1) When an application is downloaded the very first time, variables will get their initial data type values, even though they have been declared with the attribute `coldretain`, and, that the controller has done a cold restart. Hence, no variables can receive their `coldretain` values before they have been stored in the aspect directory. Correspondingly, will variables that have been declared later on, contain their initial values until they have been saved in the aspect directory.

Table 8. Parameter attributes

Name	Description
no attribute	The parameter value is not maintained after a restart, or a download of changes. Instead, it is set to the initial parameter value. If the parameter has no initial value assigned, it will be assigned the default data type value, see Table 6 on page 86 . ⁽¹⁾
retain	The parameter value is maintained after a warm restart, but not after a cold restart. ⁽¹⁾
coldretain	The parameter value is saved in the aspect directory, and retained after warm or cold restart. ⁽¹⁾ Coldretain overrides the retain attributes in a structured data type.
hidden	The parameter will be hidden for an OPC client connected to an OPC server for AC 800M. This attribute is used for variable values not necessary to a supervisory system.
by_ref	This attribute is used for controlling the passed value. For in and out parameters the value is usually copied into the called instance at the invocation. But for non simple data types and strings it is time consuming. In that case, a reference to the data instance is passed in the function block call. This is achieved by setting the attribute of the parameter to by_ref.

(1) These attributes are valid if the parameter is not connected, if connected it is the attributes of connected variables.



In case of power failure, SIL3 applications are restarted using cold retain marked values which are periodically saved in the controller with a cycle time set by the user.



Coldretain is not allowed in SIL3 application on restricted parameters. This could lead to a failing coldretain save and that the controller shuts down after a power fail restart. A system alarm is generated if coldretain fails and the controller log gives information on the problematic POU and variable.



It is possible to assign several attributes to a variable for example, retain, nosort, and hidden can be assigned as (retain nosort hidden) attribute.



An intermediate variable (a variable which is automatically generated when making a graphical connection between function blocks) in FBD or LD is always assigned the attribute retain (even if the parameters on both sides of the graphical connection have the attributes coldretain).

In addition to the general attributes, the data type editor supports two special attributes for the components (see [Table 9](#)).

Table 9. Special attributes for data types

Name	Description
displayvalue	<p>This attribute is applicable for a component in a structured data type, which is used in the FD code block of a diagram. In the FD code block, the online value label is not shown for graphical connections of structured data type. However, if one of the components of the structured data type is marked with the <i>displayvalue</i> attribute, the online value label is shown for this component.</p> <p>Note: The <i>displayvalue</i> attribute can be assigned to only one component in a structured data type. It also possible to assign this attribute through a combination of other attributes (for example, <i>coldretain displayvalue</i>, <i>nosort retain displayvalue</i>, and so on). In this case also, only one component can have this type of attribute (with <i>displayvalue</i>).</p> <p>If a nested structured variable is used, the <i>displayvalue</i> attribute is needed on each level down to the value that should be shown.</p>

Table 9. Special attributes for data types

reverse	<p>The <i>reverse</i> attribute can be used while declaring sub elements in a structured data type. This attribute is used when declaring structured data types that are intended to represent a connection between different program organization units in the applications. The signals in the connection can pass data in different directions, and the <i>reverse</i> attribute specifies that the direction of this signal (component) is <i>opposite</i> to the normal flow (forward or backward).</p> <p>It is an error if a sub element of a structured data type has the <i>reverse</i> attribute and the sub element itself is of a structured data type containing a <i>reverse</i> attribute at any sub level in that sub element.</p> <p>The <i>reverse</i> attribute affects the usage of control module parameters, diagram parameters and communication variables.</p> <p>Note: All components that have the <i>reverse</i> attribute must be placed consecutively in the data type editor.</p>
	<p>Restrictions with reverse attribute for Split and Join blocks in FD code block</p> <p>In the FD code block in Diagram and Diagram Type POU's, the Split and Join blocks must not be used for structured data types with <i>reverse</i> components.</p> <p>Restrictions with reverse attribute for function blocks</p> <p>If structured data type is used for communication between function blocks, ensure that the type does not contain any <i>reverse</i> components.</p> <p>Restrictions with reverse attribute for data exchange between control modules or diagrams using Access Variables</p> <p>It is also not possible to write to <i>out</i> direction variables using Access Variables in Structured Text, even if the attribute is <i>reverse</i>.</p>

Attribute Example

The following example tries to illustrate how a variable will be handled, depending on different attribute settings. Suppose the variable valveC has the attribute coldretain, valveR has the attribute retain and valve has no attribute. Also, suppose that these three variables have the initial value = True (see [Figure 28](#) for the variable declaration).

	Name	Data Type	Attributes	Initial Value
1	valveC	bool	coldretain	true
2	valveR	bool	retain	true
3	valve	bool		true
4				

Figure 28. Three variables with different attributes settings

According to the attribute settings in [Figure 28](#), the variables will be read or written on different occasions in the given code example below, (read the comments under each IF statement):

```

IF valveC THEN
  (*Code in this position is only executed once after the very
  first cold restart*)
  valveC := false;
END_IF

IF valveR THEN
  (*Code in this position is only executed once after a cold
  restart*)
  valveR := false;
END_IF

IF valve THEN
  (*Code in this position is only executed once after a cold restart
  and once after a warm restart*)
  valve := false;
END_IF

```

Note that execution does not have to take place during the first scan after restart, for example, when IF valve is embedded in another IF statement.

Variables and parameters should have the attribute retain, unless they are written at each scan. When a change has been made to the application, the entire application will be (warm) restarted and in doing so, variables without the attribute retain will be set to their initial values, and there is a chance that the change will not be totally bumpless. It is recommended that In and Out parameters to function blocks always have the attribute retain.



More information is given in Control Builder online help. Search the index for “attribute”.

Initial Values

It is possible to give the variable an initial value, which will be assigned to the variable the first time the application is executed. This setting overrides the default data type value. [Table 6](#) shows default initial values for the most common data types.

Descriptions

The description field describes and provides information about the variable. A short descriptive text may include an explanation of the cause of a condition or a simple event, for example “Pump 1 is running”. Since the description is not downloaded to the controller, the size of the description is irrelevant.

Specific Initial Values

In the Control Properties aspect, the user has the possibility to set instance-specific initial values for variables and parameters in a POU that are different from the ones defined for the type. These values are compiled and applied to the instances when Control Builder downloads the project to the controllers. Specific initial values can be set for the following types of objects in the Control Structure:

- Applications (for variables and global variables),
- Program (for variables)
- Single control modules (for variables and parameters that are default-marked and not connected)
- Control modules (for variables and parameters that are default-marked and not connected)

- Function blocks (for variables and parameters with direction in or out, but not for the direction in_out)
- Diagrams and instances of diagram types

Set Specific Initial Values

Specific initial values are set in Plant Explorer, via the Control Properties aspect in the Control Structure. To enter an initial value:

1. Select the Control Properties aspect for the object.
2. Select the Properties tab.
3. Select the corresponding item with the Init_Val suffix, then enter the initial start value in the Property Value field.
4. Click Apply.

If Control Builder finds errors when compiling instance-specific initial values before download, Control Builder presents a dialog where errors can be corrected.

Priority Order

Initial values are applied in the following order:

1. Coldretain value from the latest saved set.
2. Instance-specific initial value (init_Val property).
3. Initial value declared in the type.
4. Default value of the data type.

Retain Attributes—Effect on Initial Values

The retain attribute decides how initial values are applied.

Table 10. Application of initial values, depending on retain attributes.

Attribute	Situation	Initial Value Applied (_Init_Val)
No attribute	Cold restart download	Yes
	Warm restart download	Yes
Retain attribute	Cold restart download	Yes
	Warm restart download	No
ColdRetain attribute	Init_Val will be applied at the very first download. For all other situations, Init_Val will not apply if there are saved coldretain values.	

External Variables

External variables are not really variables, in the sense that they carry a value. Instead, external variables work like parameters, that is, they point to a variable value (in this case a global variable). In order for an object to reach a global variable (located at the top of the application) it must use a pointer, or more specifically, an external variable. By declaring an external variable inside an object, it is possible to access global variables efficiently from a deep code design, without having to pass variable values through parameters.

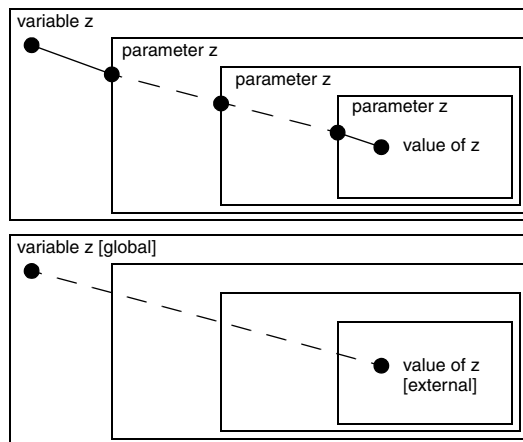


Figure 29. The variable *z* can be accessed deep down in the structure, using several parameters. (Bottom): Using external (and global) variables, the variable *z* is accessed directly, without having to use parameters.

Access Variables

Access variables are needed when the system works as a server. Allowed protocols are MMS, COMLI, MODBUS TCP and SattBus. MMS and SattBus variables are declared in the Access Variable Editor under the corresponding tab, COMLI and MODBUS TCP variables under the Address tab. The variable name must be unique within the physical control system.

Open the Access Variable Editor by right-clicking the 'Access Variables' icon under the respective Controller and select **Editor**.



To limit the access to a variable, set the attribute to `ReadOnly`. If the attribute is left blank, it is possible to both read and write.



Set the Attribute to 'ReadOnly' if the connected variable is located in a SIL application. A compile error is shown if the `ReadOnly` attribute is not set. And for the Address tab (that has no attribute column) it is allowed to connect the access variable to a SIL variable, but the variable is implicitly `ReadOnly`. A client that tries to write to the access variable will get a runtime error (or a controller shutdown if the controller is pre SV6.0).

MMS

MMS variables can only be accessed by name.

An MMS access variable name can be up to 32 characters long and contain letters, digits and the characters dollar(\$) and underscore(_). However, an access variable name cannot begin with a digit or the dollar (\$) character.

All data types for single and structured variables are allowed, with the exception of ArrayObject and QueueObject.

To limit the access to an MMS variable, set the Attribute to ReadOnly. If the attribute is left blank, both read and write is possible.

SattBus

SattBus variables can be accessed in three ways:

- Standard SattBus name such as Valve:
 - the name must consist of exactly five ASCII characters, but may not begin with a percentage sign (%).
- COMLI direct addressing (see [Address](#)),
- IEC 61131-3 standard representation for variables.
 - IEC61131-3 address must be entered under the COMLI tab

Allowed data types for a single variable are, bool, dint, int, uint, real or string. Whereas a structured variable does not allow string data type.

Address

Address variables can be accessed in two ways only, either direct addressing with capital X and the number for boolean, or capital R and the number for registers (R0-R65535 for PA controller and 65000 for HI controller) beginning with a percentage sign or not, or according to IEC 61131-3 standard representation for variables.

Allowed data types for a single variable are bool, dint, int, or uint, whereas structured variables must all be of same data type. A structured variable is allowed to contain more than 512 booleans and contain more than 32 components of integer data type. Overlapping areas are not allowed.

Example

An access variable name "X0" is defined and connected to a variable which contains 544 Boolean components at octal address 0-1037. The next available address is then 1040 to ensure that areas do not overlap.

At least one of the variables in the access variable table has to be defined. For missing variables, requested data of boolean data type will be returned with the value False and requested data of integer data type will be returned with the value "0". Writing to undefined variables is ignored.

Communication between Applications Using Access Variables

Two applications may communicate with each other via variables, but these variables must be declared as access variables (see, [Access Variables](#) on page 97). This also applies when two applications are downloaded to the same controller (see [Figure 30](#)).

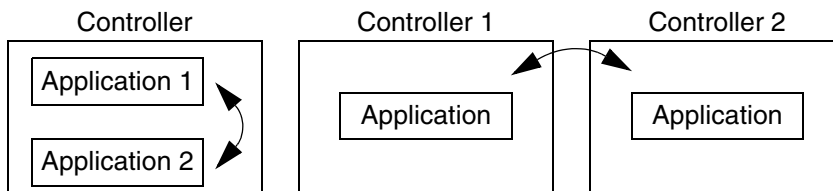


Figure 30. Variables for communication between applications must always be declared as access variables.

When transferring access variables, it is important to use the same data type range for the client (*dint*), as for the server (*dint*).

It is, however, possible to connect variables with different ranges, such as a *dint* variable on the server and an *integer* variable on the client.

As long as the variable values are within the range of an integer, this will work, but once the value goes outside the integer range, it will not.



If an access variable is the only user of a variable that is connected to an I/O channel, this variable is by default updated every second. To update this variable with another interval, create a statement that involves the variable, but is never executed.

A statement that is never executed, but still updates the variable x could look like this:

```
if false then
    x:=x;
end_if;
```

Connect this program to a task that executes with the desired interval. The variable is updated every time the task is executed.

Communication in an Application Using Global Variables

In Programs

Global variables are declared at application level, in the Global Variables tab of the application editor. They can be accessed directly, without any declaration in the program editor. Variables that are not declared in the declaration pane in the program editor are assumed to be global variables. A global variable can be used in any program, without having external variables declared in a program.

In Function Blocks or Control Modules

In order to reach a global variable from either a function block type or a control module type, each type must have either an external variable declared or a parameter. Thus, the types access the global variable value by using an external variable or a parameter to point at the global variable located in the application.

Communication Variables

The communication variables are used for cyclic communication between top level diagrams, top level single control modules, and programs, in the system network that uses MMS communication protocol.



Communication variables can be used in SIL 1-2, SIL3, and Non-SIL configurations.

Communication variables are declared in the Program editor, top level Single Control Module editor, or top level Diagram editor. Communication variables support both inter application communication and inter controller communication in a system network.



Communication variables are not supported in distributed applications. If an application that contains communication variables is running in a controller, it is not possible to download the same application to another controller.



It is possible to change the placement of the variable port for a communication variable reference object in a diagram, from left to right and vice versa. This is needed to read data from an out-variable.

A communication variable can be either a communication input variable or a communication output variable.

If the direction of a communication variable is *in* in a POU, the POU can read the variable, but cannot write to the variable. If the direction of a communication variable is *out* in a POU, the POU can write to the variable and read the variable.

A communication variable can be either an elementary type or a structured data type. It cannot be a generic or built-in type.



If a communication variable is of structured data type, it must not contain components that are declared with the **CONSTANT** type qualifier and it must not contain **CONSTANT** components at any sub-level of the variable.

Communication variables use a name based resolution to connect a communication output variable to one or several communication input variables.

In a system network with Non-SIL configuration, all communication output variables must be declared with unique names.

In a system network with SIL1-2 or SIL3 configuration, all communication output variables must be declared with unique names and unique IDs.

Communication variables cannot be connected to the channels of an I/O unit. Therefore an application code has to be entered to transfer values between communication variables and local variables, which are connected to I/O.

Declaration pane for communication variable

The declaration pane for communication variables consists of:

- Name

The name of the communication variable. For communication output variables (direction - `out`), the name must be unique on the network to resolve the IP-address during compilation.

- Data Type

The supported simple data types are `Bool`, `Dint`, `Uint`, `Int`, `Dword`, `Word`, `Real`, and `String`. The data types `Time` and `Date_and_time` are also supported. The string data type is not used in SIL3 communication and it is not allowed to declare a communication variable of type string in SIL3 or with Expected SIL set to SIL3.

Structured data types having components of simple data types are also supported, with maximum size of 1000 components or 1400 bytes for non-SIL communication, and 78 bytes for SIL communication. Each component occupies different size depending on type (`bool` 1 byte, `uint int word` 2 bytes, `dint dword real` 4 bytes). In SIL, all components of the structured data type must have a configured ISP value.

Communication variables can consist of structured data types that are nested in several levels.

- Attributes

Possible attributes to specify are:

- `retain`
- `coldretain`
- `hidden`
- `hidden retain`
- `hidden coldretain`

If no attribute is specified when the communication variable is declared, `retain` is filled in automatically by the editor.

- Direction

The possible values are `in` or `out`. If no direction is specified when the communication variable is declared, `in` is automatically filled in by the editor.

- Initial Value

An initial value is assigned to the variable when the application is executed first time. This setting overrides the default data type value.

In a SIL application, ISP Values are set initially instead of the Initial Value.

- ISP Value

Applicable only to communication input variables. This field defines the ISP (Input Set as Predetermined) value to be set for the `in` variable. This value can only be set for simple data types. For non-SIL applications, if no ISP value is specified, the default value is the last good value, or if no last good value exists (because of no communication), the init value is applied.

ISP values are mandatory in SIL applications.

For structured data types, the ISP values can only be set in the data type for each individual component (in the Data Type editor). Hence, it is not possible to configure instance specific ISP values for structured data types.

ISP could be used in a structured variable to detect communication failure or bad quality, by using a Boolean Valid component with ISP set to false.

- Interval Time

Communication cycle time for peer-to-peer communication. The possible values are fast, normal, slow, very fast, and very slow. The default value is normal.

The time interval (in milliseconds) for each of these cycle times is defined in the hardware editor for IAC MMS in the Control Builder. The IAC MMS object is located at position 5.1 under the controller object in the hardware tree in Control Builder.

- IP Address

Applicable to communication input variables, and also applicable for communication output variables if bidirectional. This field defines the IP address of the controller that contains the corresponding communication output variable (with the same name) in any of its applications.

When no value for the IP address is entered, the editor automatically fills in the default value `auto`. This means that the IP address is resolved during compilation.

However, it is not possible to resolve the IP-address during compilation if the in- and out-variable resides in different 800xA systems in the network. The IP-address must then be entered manually in this column.

- Unique ID

Applicable to SIL applications. It is also applicable in a non-SIL application, when reading data from higher SIL. Unique ID is an integer (32-bit unique identifier) that logically connects an in variable to an out variable. Ensure that the value of the UniqueID is unique on the entire network. An in variable with a certain UniqueID can only read from an out variable with the same UniqueID. This field therefore provides an additional safety feature, apart from the unique name of the communication variable. The default value for Unique ID is 0. This value is not accepted for SIL communication, and valid value for the unique ID must be set.

Even if the in variable is located in a SIL3 or SIL2 application and the out variable is located in a non-SIL application, the Unique ID must be specified for the SIL3 or SIL1-2 communication variable.

Even if the in variable is located in a non-SIL application and the out variable is located in a SIL3 or SIL2 application, the Unique ID must be specified at both ends.

- ExpectedSIL

Applicable to communication input variables, and also applicable for output variables if bidirectional. ExpectedSIL specifies the expected SIL of the server application that holds the output communication variable. The client checks if the ExpectedSIL matches with the SIL in the received response (if included). The following values can be selected for ExpectedSIL:

- **Same** - can only be used if client and server have the same SIL. This is the default value.
- **Non-SIL** - used if in variable is located in SIL2 or SIL3 and out variable is in non-SIL.
- **SIL2** - used if in variable is located in non-SIL or SIL3 and out variable is in SIL2.
- **SIL3** - used if in variable is located in non-SIL or SIL2 and out variable is in SIL3.

- Acknowledge Group

Applicable to communication input variables, and also applicable for output variables if bidirectional. Acknowledge group is used to categorize the communication variables in different groups for acknowledgement purpose after their ISP values get latched. This avoids unexpected restart of communication after fault detection. The possible settings are *auto* or a group ID.

- **auto** - The communication resumes automatically after the error situation is resolved. This is the default value for non-SIL applications.
- Specifying a **group ID** - This enables the communication variable for an acknowledgement after fault detection. For SIL1-2 or SIL3 communication, the default value is zero, which is not allowed in a SIL application. Therefore, it is mandatory to configure the Acknowledge Group to a value (either *auto* or a specific group ID). If a group ID is specified, the acknowledgment is performed through the CVAckISP control module, for a particular group or cascaded groups. The CVAckISP control module is available in BasicLib.

A maximum of 32 Communication Variables can be grouped together with the same group ID per application. This is also checked during application compilation.

One control module instance of CVAckISP is used to reset all the latches in one group of communication variables. If several such groups are to be reset simultaneously, the control module instances of CVAckISP for each group may be interconnected in a cascade configuration. The reset order is distributed to all members in the configuration.

- Description

User documentation of the variable.

Source and Sink for Communication Variables

The term 'source' is used for the POU that declares a communication output variable. The term 'sink' is used for the POU that declares a communication input variable.

If a sink is located in one application, a source can be located in any of the following:

- In the same application as the sink.
- In another application but in the same controller as the sink.
- In another application and in another controller.

Multiple sinks can be linked to the same source.

For example, for every communication output variable with a unique name, there can be multiple communication input variables with the same name as the communication output variable. The communication input variables can reside in a different POU, in a different application, or in a different controller.

There is no need to declare the location of the source (communication output variable) while configuring the sink (communication input variable). This is because the binding between them is based on the name of the communication variable.

The Control Builder checks whether the name of a communication output variable is unique in the 800xA System, only during the download of the application. The download is aborted if the variable name is not unique.

Unresolved Communication Variable

A communication input variable is unresolved if there is no communication output variable (source) with the same name, during compilation.



The Control Builder allows the execution of an application that contains unresolved communication variable. When a re-configuration of the system is done (for example, at a warm restart), the source can be created and the unresolved communication variable becomes resolved.

A resolved communication variable does not become unresolved if the source is removed. It gets unresolved the next time, when that particular application is reconfigured.

Bidirectional Communication Variable

Bidirectional communication variables have communication in both directions and can be configured for one-to-one connections only. These variables can be created for structured data types only.

The configuration parameters that are used for the `in` variables can also be specified for the `out` variables, if bidirectional. This allows the configuration of a communication variable with a different communication setup in either directions (for example, different interval times).

Reverse attribute

For bidirectional communication using structured data types, a `reverse` attribute must be set to indicate which components communicate in the opposite direction to the `in/out` declaration of the communication variable.



The `reverse` attribute is configured in the data type editor.

The `reverse` attribute can only be set such that all `in` variables are located consecutively and also all `out` variables are located consecutively in memory. Hence, it is not possible to configure `reverse` for every other component in a data type.

The reverse attribute can be set in both top level and sublevel of a structured data type, but cannot be nested.

This means, the reverse attribute cannot be set for a structured data type component *Struct2* inside a structured data type *Struct1*. But, *Struct2* can have reverse components inside it.

For example, for a `ControlConnection` data type, which consists of one forward structure and one backward structure, the reverse attribute is set on the whole backward structure. All components in the backward structure inherits the reverse attribute automatically.

Interval Time

Out of the five different cyclic categories (`VerySlow`, `Slow`, `Normal`, `Fast`, `VeryFast`), the default interval time for a communication variable is `Normal`.

The interval time for a communication variable can be changed only when the Control Builder is offline. The changes takes effect during the download.

The time interval (in milliseconds) for each cyclic category is defined using the hardware editor for IAC MMS. The IAC MMS object is available at position 0.5.1 under the controller object. Position 5 contains the IP object.

Hardware Simulation with Communication Variables

It is possible to use hardware simulation for IAC, except when the client is a real (non-simulated) HI controller.

A HI controller, which is a non-simulated IAC client, only accepts data from a non-simulated server and that is not a soft controller. If a server is found to be simulated, ISP is set for the communication variable.

In a PA controller, which is an IAC client, the data from a simulated server is copied in to the application, but the status of the communication variable shows that the server is simulated.

When using hardware simulation, the communication variables use real communication and real copying of input variables. This is also the case when downloading a simulated AC 800M to a Soft Controller.

Application Download

The communication variable configuration is downloaded when the application is downloaded to the controller. It is possible to download an unresolved *In* (or bidirectional) communication variable, even though the communication will not happen. To resolve an unresolved communication variable that already exists in a controller, the new configuration with the Out variable must be downloaded.

When an out variable is removed, only *in* variables that are defined in applications which are downloaded, shows communication variables as unresolved. Other communication variables (in other applications) will timeout.

To support multi user engineering, all the affected controllers are reserved during last step of the communication variable analysis, until the configuration is downloaded.

Communication from Lower SIL to Higher SIL using Diagrams

For communication between different applications, the only way to use a signal from a server application with lower SIL (lower than the SIL of the client application) is by using a graphically connected communication variable reference in the FD code block of a top level diagram. This means that the FD code block in a diagram is the only code block in Control Builder that supports a lower SIL signal input.

Compared to an ordinary communication variable reference, there are two differences for this type of communication variable:

- It is displayed in yellow.
- The Expected SIL value is also displayed as a label below the object.

The indication is shown in both Offline and Online modes.

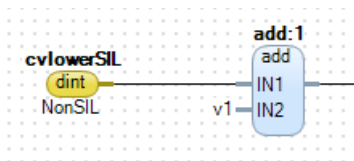


Figure 31. Communication variable with lower Expected SIL displayed in yellow

Communication Variable Limits Dialog

This dialog enables to modify compiler settings related to restrictions of Communication Variable usage. Select **Settings - Communication Variable Limits** in the context menu of the Project object to open the dialog. The following dialog is displayed.

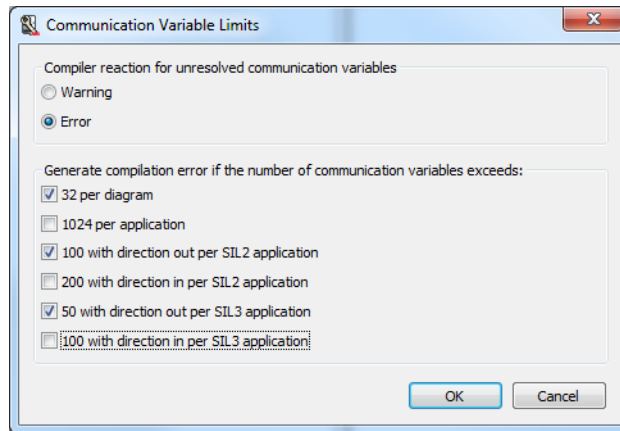


Figure 32. Communication Variable Limits Dialog

The dialog consist of two sections:

- *Compiler reaction for unresolved communication variables*
- *Generate compilation error if the number of communication variables exceeds*

The *Compiler reaction for unresolved communication variables* settings is related to a compiler check. The check cannot be turned off, but can only choose between compilation warning or compilation error. See [Unresolved Communication Variable](#) on page 107 for information about unresolved variables.

All the settings in the *Generate compilation error if the number of communication variables exceeds* section are related to different compiler checks. Each check can be turned off separately. Compiler errors are generated if the settings are enabled and the number of Communication Variables¹ exceeds:

- 32 per diagram
- 1024 per application

1. This is the sum of internal and external Communication Variables, see [Communication Variables](#) on page 343. A Communication Variable may include one or several components.

Example of the generated Communication Variable compiler error:

```
Too many communication variables in application (Max: 1024  
Used: X)
```



The purpose of the limit of Communication Variables per diagram is to guide the user to avoid large and complex diagrams.



Exceeding 1024 Communication Variables per application might risk the general execution stability of the controller, and it is not recommended to disable this check.

For SIL applications there are additional configurable checks in the dialog for the number of Communication Variables:

- 100 with direction out per SIL2 application
- 200 with direction in per SIL2 application
- 50 with direction out per SIL3 application
- 100 with direction in per SIL3 application

Communication Variables communicating between different IEC 61131-3 SIL tasks (same or different controllers) affects the IEC 61131-3 execution time, due to the IAC fast data copying. If the number of Communication Variables checks per SIL application is exceeded, this means that the IAC copy makes up a significant part (more than 10%) of the recommended maximum IEC 61131-3 execution time 100 ms. Note that higher prioritized IEC 61131-3 tasks cannot interrupt a lower prioritized IEC 61131-3 task during IAC fast data copy.



If the impact on the IEC 61131-3 execution time due to a high number of SIL Communication Variables is per design of the project, and a proper task tuning using the Task Analysis tool is performed, then the compiler checks of Communication Variables per SIL application can be disabled.

Control the Execution of Individual Objects

Sometimes there is a need to execute specific sub function blocks and/or sub control modules, with a time interval and priority different from the task connected to the application. Depending on the requirement, this can be done in two ways:

1. To create a new task and connect this task to all the following objects, read the sub-section '[Using a Global Variable Connected to an External Variable](#) on page 113.
2. To choose a new task for each individual object (and for that object only), read the sub-section '[Using a Global Variable Connected to a Parameter](#) on page 114.

Using a Global Variable Connected to an External Variable

Assume that the user has added a new task, for example **SuperFast**, to the other tasks in the Project Explorer.

Steps to use global variable:

1. Declare a global variable (for example *Speed*) of data type *string*, with the attribute *constant* and the initial value '*SuperFast*'.
2. To reach objects that have been created in the application, start by declaring an external variable in the type (open the type editor and select the external variable tab).
3. Declare an external variable with the same name, data type and attribute as the global variable. In this example, an external variable called *Speed* of data type *string* and with the attribute *constant* is used.

Finally, connect the new task **SuperFast** to the object by right-clicking the object and selecting **Task connection**. Type the variable name *Speed* in the task field. All the following objects that are created will have this task connection, that is, **SuperFast**.

The advantages with this method of using a global variable connected to an external variable (declared in the type) is that every following object will be connected to the same task (SuperFast). If the user later on need to change the task connection for all the objects (perhaps hundreds of objects), change only the initial value for the global variable in the application (see [Figure 33](#)). The present task connection for all

objects will point, via the external variable to the task declared by the global variable.

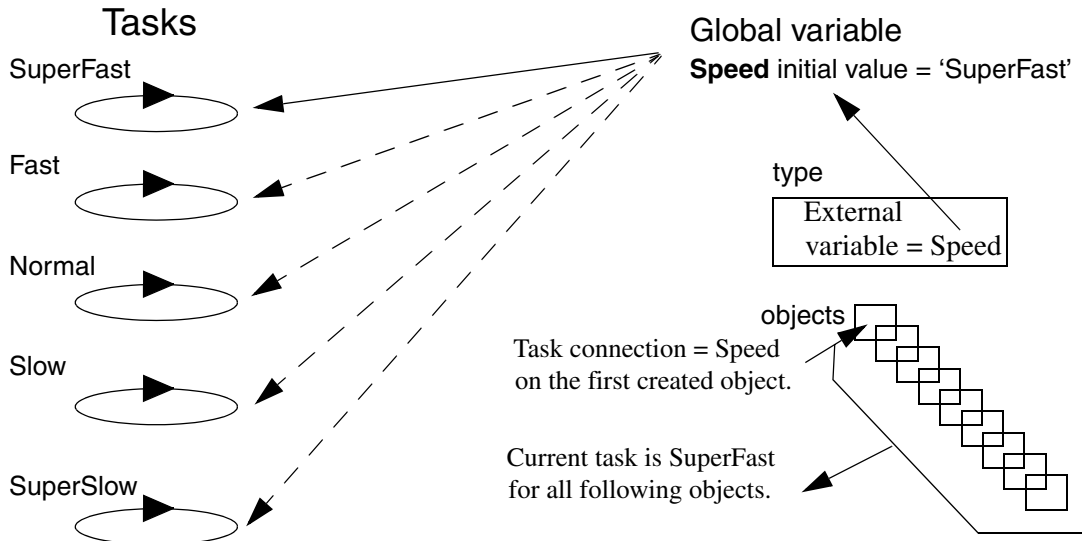


Figure 33. All objects will have the same task connected (*SuperFast*), once the first object has connected *Speed*.

Using a Global Variable Connected to a Parameter

Assume that the user has added a new task, for example **SuperSlow**, to the other tasks in the Project Explorer.

The main advantage of this method, compared to the previous method with external variables, is that the user can change the task connection on each following formal instance, by simply connecting a parameter to a different global variable. (See [Figure 34](#)).



For more information on formal instances, see [Types and Instances - Concept](#) on page 44.

This method is based on declaring two global variables (for example, *Slowly* and *Learning*) of the data type *string*, with the attribute *constant*, and the initial values '**SuperSlow**' and '**Slow**', respectively.

In order to reach the following objects that have been created in the application, start by declaring a parameter in the type (open the type editor and select the parameter tab). Declare a parameter, for example *Sleepy*, of data type *string*. Select the formal instance (object) inside the type:

1. Right-click the object and select **Property > Task connection**.
2. Type *Sleepy* in the task field.

Every created object that is based on the type (containing the formal instance) can be connected via the connection parameter *Sleepy* and one of the global variables *Slowly* or *Learning*, located in the application.

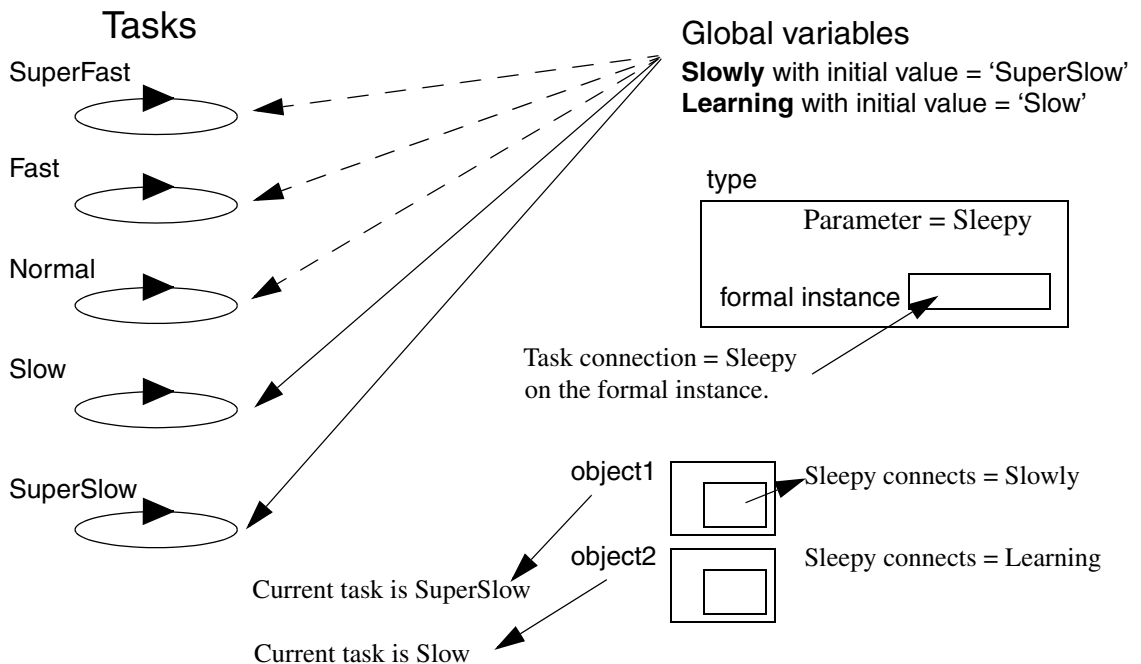


Figure 34. Each object can be connected to a different task via the parameter *Sleepy* declared in the type and task connected in the formal instance.

The advantage of this method is that the objects of the formal instance, located inside the type can be connected to different tasks (global variables with a different task name as init value).

Link Variables in Diagrams

For a graphical data connection in a diagram editor, it is possible to set a link variable. A local variable of the same data type as the connected ports can be used.

To create a link variable, right-click the required graphical connection to open the context menu, and select **Link Variable**. Enter the name of the link variable.

The link variable name is not displayed in the graphics; it is only visible if you open the Link Variable dialog.

Normally, link variables are not needed. The compiler auto-generates variables when necessary. But, if it is needed to access the intermediate value via OPC, a link variable is required. The declared link variables are available via OPC.

If an output port has multiple graphical connections, all these connections share the same link variable (if any).

If a block has an *in_out* port that is graphically connected on both sides, both connections share the same link variable (if any). This means that if several blocks have graphically interconnected *in_out* ports and a link variable is set for one of the graphical connections, the link variable is forwarded and set to all the graphical connections in the *in_out* chain.

Project Constants

Project constants are declared at the top level of libraries and projects. They are globally visible, and can be used wherever a constant value is permitted, for example, in program code and for variable initialization. With project constants, it is possible to create settings for an individual project, without having to modify any source code, or having to introduce parameters which have to be passed on to all concerned types.

Project constants are suitable to use for library items that the user wants to change. Examples are, date and time formats, logical colors and logical names. Do not use

project constants to change the functionality of an object, for example, initial values and comparisons in code.

Typically, project constants are declared in a library and given default values. They are then used, for example, in code located inside types.

Project constants are allowed to have the same names as variables and parameters. Control Builder will, however, choose the variable or parameter name if a name conflict exists. This must be considered when adding, renaming or deleting variables or parameters in an already running application.



Follow the naming convention, which says that project constants should begin with the letter “c” (for example “cColors”). Use structured project constants, if possible.



Note that project constants cannot be used to control the execution of function blocks or control modules. Use a global variable or a parameter instead. For more information see, [Control the Execution of Individual Objects](#) on page 113.



If a project constant connected to a retain parameter (or variable) is changed online, then the change does not effect on existing instances until a cold restart is performed.

Project constants declared at library level (user-defined libraries) can only be edited and deleted from the library, that is, they cannot be deleted from the Project constant dialog that is reach by right-click the control project folder (root object). To edit or delete a library-declared project constant, right-click the library in Project Explorer and select **Project Constants**.



Naming conflicts between project constants appears when the same project constant name exists in more than one library at the same time.

The only way to avoid a naming conflict is either to delete one of the constants or not using the constant at all. A type conflict can never be overridden.

Structured Project Constants

It is advisable to create one single structured project constant for an entire project or library, where the project constant name is a concatenation of “c” and the project name (or library name).

An example:

If the project name is “ACMEToothpaste”, the structured project constant should be named “cACMEToothpaste”. Using a structured project constant makes sure that there is little chance of conflict with variable and parameter names. Using a structured project constant (“cACMEToothpaste”) enables the user to, for example, use “Max” without causing problems due to a variable or parameter called “Max”, since the full path to the project constant “Max” would be “cACMEToothpaste.Max”.

Define only one project constant per library. This project constant can, and should, be a structured project constant the concatenation of “c” and the library name in which it is contained. For example, if the library name is “ACMEValveLib” the (structured) project constant should be “cACMEValveLib”.



All project constants defined in libraries and projects must have been given unique names.

Typical Use

There are two typical use cases for project constants:

1. To satisfy the need for constant values in all project applications.

Some values might have to be constant throughout the entire project. To change such a “constant” value, change it once. There is no need to change it at every occurrence. For such cases, use a project constant. The project constant is defined in one place only, and can be used throughout the project. Changes to the project constant will be reflected throughout the project.

An example:

To be able to change the severity for all “High level alarms” in the entire project, set up a project constant that defines the severity and use the project constant in all alarm blocks in all applications. To change the severity, just change the value of the project constant.

In this case, project constants should be defined on control project level, not in a library.

2. To be able to change library type solutions without having to make changes in the library itself.

A method commonly used in control application engineering/programming is to construct libraries, in which re-usable code is placed. It is good practice to make the library as general as possible, to maximize its usefulness. The use of project constants is an excellent solution for such situations.

Example 1: Easy Translation

Assume that the user has created a library that makes extensive use of text strings. Instead of including strings (in the user's native language) statically in the library, use project constants. This allows another engineer to change the values of these project constants and to translate the strings to another language.

For example, a project constant that was originally set to "Stop" can easily be translated by a German engineer to "Halt", simply by changing the value of the project constant. This would not be the case if the user had typed "Stop" in the library. Such string constants that are to be translated are best stored as a structured project constant under the component *.Settings*.

The string "Stop" would, for example, be defined as the structured project constant "cACMEValveLib.Settings.StopLabel" or, even more levels; "cACMEValveLib.Settings.Labels.Stop".

Example 2: Combination of Dynamic and Static String Constants

Consider the following function block, in [Figure 35](#), that controls high alarms. *Signal* is of *RealIO* type, *Alarmlevel* is of *real* type, and *Message* is of *string* type.

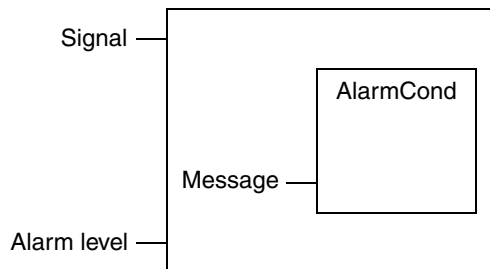


Figure 35. The function block *AlarmCond* located in the *Alarm* library.

Now, we want a "customized" message to be passed to *Message*, such as

High Level (> 75 °C)

The message consists of five important elements that make up the message.

1. “High Level”
2. “(> “(note the spaces)
3. 75 (a value set by Alarm level)
4. °C (a value set by `Signal.parameters.unit`)
5. “)”

All in all, three strings (1, 2, and 5) and two values (3 and 4).

Defining these 3 strings locally would be poor design, since the strings would be defined for every object that is created from the type. To create a dynamic environment, use project constants, or, more specifically, structured project constants.

In the example above, we actually have different string categories – “**High Level**”, “(> “, and, “)”

The first one is a (dynamic) string that a user may want to translate, depending on target customer nationality, whereas the other two are static and independent of language. This calls for two different views of project constant.

Using structured project constants, and the naming convention mentioned earlier in this section, a defined structured project constant for “High Level” could be: `cACMEValveLib.Settings.HighLevelLabel`.

As described in the first example (Example 1 above), we make use of the component “Settings” in the structure. Underneath this component, we define the constants that are to be translated, or changed, depending on circumstances.

Next, we define the structured project constant `cACMEValveLib.Internal.Str1` and `cACMEValveLib.Internal.Str2` to contain “(> “and “)”. Note the component “Internal”, which implies that components (constants) under this level are not to be changed by the user. Of course, the user can use the structure `cACMEValveLib.Settings.Labels.HighLevel`, as described earlier, if the user prefers more levels.

I/O Addressing Guidelines

A good I/O variable structure is the key to being able to debug and change an application. A good structure also makes the connection of the application to system I/O easier to read and understand.

Below are some hints and tips to ensure that the I/O connections have a good structure.

- A good I/O connection structure requires a good application program structure, and also a realistic translation of the process to be controlled, into the application program.
- Try to collect I/O of the same process object in the same controller, and even in the same object in the application program.
- Try to divide the application program into process cells, with contents similar to the real process.

These hints are basic rules for object-based programming for real processes, and once the application has a good structure, it is easier to divide I/O signals into groups or cells of the process.

Connecting Variables to I/O Channels

Only one variable can be connected to each I/O signal, and vice versa. This is not a problem for output signals, but for input signals it may be necessary to read the same input signal from different programs, or even from different places in the same program. This can be done by placing the connected IO variables in a common area, for example, in the application. Then the variables can be read by the program(s).

Note that the result of an IO copying is different depending on whether the parameter is IN or IN_OUT. An IN parameter will result in a copy of the value, whereas an IN_OUT parameter will result in a reference to the current value. While different tasks can copy the same I/O signal, a task with a higher priority may update the signal value in the middle of a scan. See also [Function Block Execution](#) on page 71 and the information on connected I/O channels in a task in the *System 800xA Control AC 800M Planning (3BSE043732*)*.

If the same I/O signal must be read by different applications, the I/O copying must be done from one of the applications. The copied value can then be moved to other applications through ordinary communication services. See also [Communication between Applications Using Access Variables](#) on page 99.

The address for a hardware unit is composed of the hardware tree position numbers of the unit and its parent units, described from left to right and separated by dots. For example, channel 1 on the I/O unit DO814 in [Figure 36](#) has the address Controller_1.0.11.1.1.

[Figure 36](#) illustrates an example of a controller hardware position.

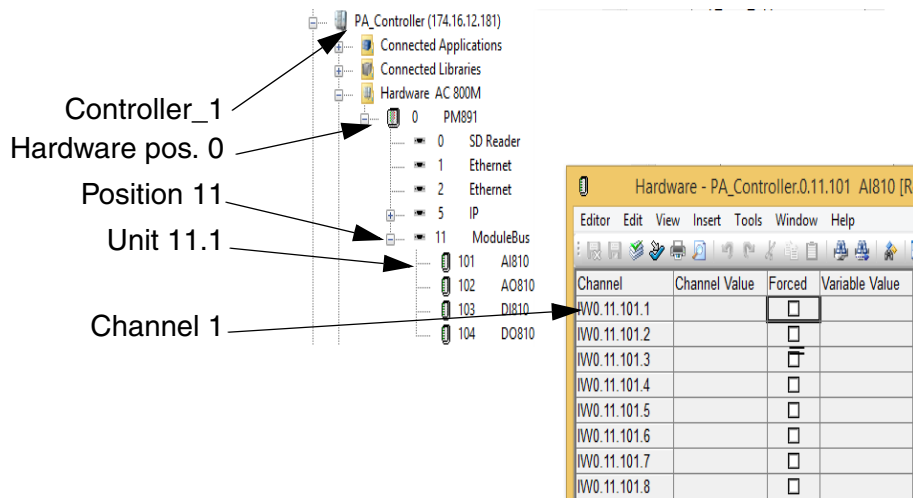


Figure 36. An example of how IO channel addresses are created in a control project.

All I/O access is done via variables connected to I/O channels and these variables are connected in the hardware configuration editor. The Connections tab displays all channels that can be connected.

I/O Data Types

Variables connected to I/O can be of any of the simple data types, *bool*, *dint*, *dword* or *real*, or any of the system-defined I/O data types. For example, an IO unit input can be connected to a variable of *bool* data type or a variable of *BoolIO* data type. For applications that only require a simple channel value, it is enough to connect a variable of simple data type. But for applications that need comprehensive information like forcing IO channels, reading status, or validate analog channel values, must connect variables that is of system defined (structured) IO data type.



It is possible to force I/O values, and display forced and non-forced values from an engineering station, regardless of whether the channel is of a simple data type or an I/O data type.



It is not possible to assign the forced component of a system defined I/O data type in a SIL certified application, but it is possible to reset a specific force using the firmware function `ResetForcedValue`.

The user can always choose a variable that is of the simple data type *bool*, *dint*, *dword*, or *real*, and connect it directly to the I/O channel, as long as the user is content with a simple value in return. However, such a connection does not take advantage of certain auxiliary signals which come with structured data types. A predefined structured data type includes signals for I/O forcing, analog signal status, maximum and minimum values, etc.



Always use `In_Out` parameters when writing to output I/O variables from a function block. This will prevent unintentional overwriting of I/O variable component values, such as scaling. Do not use `Out` parameters for this purpose.

Figure 37 presents as an example the available components inside the structured data type *BoolIO*.

	Name	Data Type	Attributes	Initial Value	ISP Value	Description
1	Value	bool	retain displayval		false	Value in the application
2	IOValue	bool	retain		false	Value from I/O before forcing
3	Forced	bool	retain		false	Tells if the input is forced or not
4	Status	dword	retain	16#00C0	16#00C0	Error status
5						

Figure 37. Components inside the structured data type *BoolIO*

A structured data type (for example, the *BoolIO* data type) contains four components. Declare a local variable *MyIOVar* as a *BoolIO* data type, and then connect *MyIOVar* to an IO channel to automatically access these four component values at the same time.



By declaring a structured data type, more information can be accessed from the IO channel, which can be read/written in code.

Declaring *MyIOVar* as a simple data type, *Bool*, provides access to the channel value. In other words, the user cannot read or write other values from the code.



When connecting a structured data type to an I/O channel, always connect the data type (like *MyIOVar*). Do not try to connect one of the components inside (like *Value*, *I/O Value*, *Forced* etc.) directly on the I/O channel.

Table 11 shows the (hardware editor) entries to different IO channels. The Type column presents the IO channel data type in the hardware editor, whereas the Variable column presents possible data type connections (simple, structured).

Table 11. Possible variable (data types) connections to IO channels.

Channel	Name	Type	Variable
IX, QX	Boolean. input (IX) and output (QX)	<i>BoolIO</i>	<i>bool, BoolIO</i>
IW, QW	Non-boolean. input (IW) and output (QW)	<i>RealIO</i>	<i>real, RealIO</i>
IW, QW	Non-boolean. input (IW) and output (QW)	<i>DintIO</i>	<i>dint, DintIO</i>
IW, QW	Non-boolean. input (IW) and output (QW)	<i>DwordIO</i>	<i>dword, DwordIO</i>
IW0, QW0	⁽¹⁾ All Inputs, All Outputs	<i>DwordIO</i>	<i>dword, DwordIO</i>
IW0	Channel status	<i>DwordIO</i>	<i>dword, DwordIO</i>
IW0	UnitStatus	<i>HWStatus</i>	<i>dint, HWStatus</i>

(1) ISP and OSP values are not set for variables connected to All Inputs/All Outputs!
For more information see also *Access All Inputs and All Outputs* on page 378.

See Figure 38 and the corresponding structured data types in Table 11.

Channel	Name	Type	Variable	I/O Description
IX0.11.2.1	Input 1	BoolIO	Application_1.Program1.MyIOVar	
IX0.11.2.2	Input 2	BoolIO		
IX0.11.2.3	Input 3	BoolIO		

Settings Connections Properties Status Unit Status

Row 1, Col 3 800xainstaller

IO channel of type boolIO.

MyIOVar of BoolIO (correct connection).

Figure 38. A correct way of connecting IO variables. The structured data type MyIOVar connected to an IO channel.

Example of I/O Channel Representation

The IO channel in [Figure 38](#), IX0.11.1.1, interpreted from [Table 11](#), gives the following: IX is a Boolean input, whereas 0.11.1 represents the hardware address and .1 represents the I/O channel.

Monitoring the Status for Hardware and I/O

UnitStatus is a hardware connection to individual hardware and I/O units in the Project Explorer. The user can connect a variable to Unit Status by selecting the Unit Status tab in the hardware editor.

If the user chooses to connect a variable to Unit Status this must be either of a dint data type or of an HWStatus structured data type. The simple data type dint will return one of the unit status value 0 (OK), 1 (Error) or 2 (Warning). Whereas, a variable of HWStatus provides more extended unit status information. See the contents inside the Unit Status tab in [Figure 39](#).

Name	Value	Description
HWState	0x2	Warning
HWStateChangeTime	2004-08-16-14:03:59	Time when error or warning occurred
ErrorsAndWarnings	0x0	
ExtendedStatus	0x0	
LatchedErrorsAndWarnings	0x100	Forced
LatchedExtendedStatus	0x0	

Figure 39. The components available inside the HWStatus.

In addition to the Unit Status there is a 'collective' hardware connection, AllUnitStatus, which contains errors and warnings regarding all hardware units connected to the controller.

Similar to Unit Status, the user can choose to connect a variable of simple data type dint or a variable of the structured data type HWStatus. The simple data type dint will return one of the unit status value 0 (OK), 1 (Error) or 2 (Warning). Whereas, a variable of HWStatus provides more extended unit status information.

Channel	Name	Type	Variable	I/O Description
IWD	AllUnitStatus	dint	ShopDoors_ST.Normal.HardwareStatus	Status of all hardware units

Figure 40. The AllUnitStatus connection gives access to the status of all units.



For information about supervising IO channels and unit status in online mode, see [Supervising Unit Status](#) on page 375.

Extensible Parameters in Function Blocks

Some function block types have extensible parameters, such as MMSRead, COMLIRead, etc. This means that the number of input/output parameters is changeable, and must be specified while declaring the function block in the function block tab.

The editor automatically inserts [1] when the user specifies a function block type with extensible parameters. Change the number within the brackets to the required number of parameters.

To see which function block types can have extensible parameters and the maximum number of parameters for each type, see the Control Builder online help.



In the Function Block Diagram (FBD) and Ladder Diagram (LD) languages, a maximum of 32 extensible parameters per function block can be shown.



There is no support for online values on Extensible Parameters. No such values will be presented in online editors or in the project documentation and consequently it is not recommended to trust these values.

Keywords for Parameter Descriptions

Types that are located in standard libraries contain keywords in the description column for parameters. These keywords help the user to organize the parameters and document the purpose of parameters.

Table 12. Type description keywords.

Keyword	Description
IN	The parameter direction is IN (read).
OUT	The parameter direction is OUT (write).
IN(OUT)	The parameter direction is both IN and OUT, but mainly IN (read).
OUT(IN)	The parameter direction is both IN and OUT, but mainly OUT (write).
NODE	Applies only to control modules. Used to indicate that the parameter has a graphical connection.

Table 12. Type description keywords.

Keyword	Description
EDIT	<p>Applies only to IN parameters. The parameter, which must have a value, is only read following changes to the application, warm restart or cold restart.</p> <p>Be careful not to connect a variable to a parameter with the keyword EDIT. Use a literal instead.</p>
NONSIL	<p>Some of the Certified Function Block Types and Control Module Types, contains SILx Restricted sub-objects.</p> <p>It is not allowed to use output parameters from Function Blocks or Control Modules marked with Non-SIL in the parameter description in a way that can influence the safety function of a SIL classified application. If such code affects an output from a SIL3 application, it might result in a Safety Shutdown.</p>

Real value in AC 800M

The AC 800M Controller stores real values according to the Institute of Electrical and Electronics Engineers, Inc. (IEEE) has standard for floating-point representations and computational results (IEEE Std 754-1985).

Floating-point consist of three fields, a sign (1-bit there 1 is positive), a biased exponent (8-bit) and a value (23-bit) gives a total of 32-bits. The range is $\pm 1.18 \cdot 10^{-38}$ to $\pm 3.4 \cdot 10^{38}$. The 24 bits (including the hidden bit) of mantissa in a 32-bit floating-point number represent **a precision of approximately 7 significant decimal digits**.

Unlike the real number system, which is continuous, a floating-point system has gaps between each number. If a number is not exactly representable, then it must be approximated by one of the nearest representable values.

Because the same numbers of bits are used to represent all normalized numbers, the smaller the exponent is, the greater is the density of representable numbers. For example, there are approximately 8,388,607 single-precision numbers between 1.0 and 2.0, while there are only about 8191 between 1023.0 and 1024.0.

How does AC 800M handle the result when result is out of limits

Arithmetical calculations near the range limits will result in either overflow or underflow.

Division by zero will result in the largest positive value if the numerator is positive and the largest negative value if the numerator is negative.

In some rare case could the result be NaN (Not a Number).

To ensure that the result is proper values use the function RealInfo. See online Help for details about RealInfo.

How to use floating point

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation. A floating point value has the type real in IEC61131-3.

Arithmetic

A floating point value has typically 7 significant figures. If a big and a small floating point value are added there is not enough significant figures.

Example:

An accumulator r1 is incremented by a small value r2: "r1:= r1 + r2;". It will work fine when r1 is small but suddenly r1 won't increase anymore, and that is because the sum has too many significant figures. If for instance $r1 = 123456.0$ and $r2 = 0.0009$ the sum r1 should be 123456.0009. But this value will be truncated to 123456.0, thus no increase of r1.

The solution is to use another accumulator that accumulates r2 up to a size that could be safely added to the final accumulator r1.

Division

Division by zero will result in the largest positive value if the numerator is positive and the largest negative value if the numerator is negative.

Comparison

Don't do "b1:= r1 = r2;" to find out if two floating point values are the same. r1 and r2 might be shown as the same value but have different binary representation. You probably want to do something like "b1:= abs(r1 - r2) < 0.001;"

Overflow

A floating point value might get the special value "overflow" if for instance two big values are multiplied. Any subsequent use of an overflow value will result in a new overflow value. Thus one overflow value might be spread through a computation unit every value is an overflow. When there is a risk that an operation might give an overflow the function RealInfo can be used. See Online Help for details about RealInfo.

Property Permissions

Parameters and variables that are not needed for HSI, configuration, etc., should have the attribute Hidden, but for all other variables that will be exposed via the OPC Server, property permissions must be properly set. Note that components inside a data type should also have property permissions. The user can set permissions from both Project Explorer and Plant Explorer.

The following five property permissions are the frequently applicable. However, there are several other property permissions available, along with self-defined property permissions.

- Read
- Operate
- Tune
- Configure
- Administrate



For more information about creating self-defined Property permissions, see the *System 800xA Administration and Security (3BSE037410*)*.

In some cases, there is also a need for setting authentication levels, besides the Read and Write property permissions.

Re-authenticate

Re-authenticate means that the user will be asked for **UserId** and **Password** before changing the property.



This function requires a separate license and is not available to all users.

Double Authenticate

Double authenticate means that two separate **UserIds** and **Passwords** have to be entered before changing the property.



This function requires a separate license and is not available to all users.

Set Property Permissions and Authentication Level

The property permissions and the authentication level can be set from both Plant Explorer and Project Explorer.

To set permissions from Project Explorer:

1. Double-click the object. The corresponding editor opens.
2. Select **Tools > Edit Permissions**. The Edit Permissions dialog is displayed.
3. Click a variable under Property, and select (Read/Write) permissions and authentication level from the drop-down menus.



The user can set the same property permission for several variables in one operation, by selecting the variables (Ctrl + mouse click) and then select permission from the drop-down menu.

4. Click **OK**.



Property permissions and authentication levels can only be set on variables and parameters of simple data type. Hence, property permissions and authentication level attributes for structured data types will display (N/A). Corresponding settings for components must be repeated inside each **Data Type**.

Property Attribute Override

Property Attribute Override is an aspect that allows the user to override existing property permissions and authentication flags on both types and objects, inside libraries. For more information, refer to the *System 800xA Administration and Security (3BSE037410*)* manual.

Library Management

From the user point of view, there are two main types of library:

- Standard libraries, that are installed with the product. These are protected and cannot be changed.
- User-defined libraries, in which users can add their own types. Copies of template types (data types, function block types, control module types, and diagram types), from the standard libraries can be modified and also added into the user-defined libraries.

The following operations are relevant to both library types:

- Libraries must be inserted into the control project in which they are used, see [Insert Libraries into Control Projects](#) on page 138.
- A library that contains types for applications must be connected to all libraries and applications that use types from the library. Libraries containing the hardware types (units) used in the controller configuration have to be connected to the controller. See [Connect Library to Application, Library or Controller](#) on page 138.
- A library can be disconnected from, an application, library or controller, see [Disconnect Libraries](#) on page 140.
- A library can be imported/exported to/from an 800xA system, see [Import/Export Libraries](#) on page 141.

The following operations are relevant to non-standard libraries only, since standard libraries are protected and cannot be changed:

- A new library can be created, see [Create Libraries](#) on page 141.
- The state of a library can be changed, see [Library States](#) on page 142.
- The version of a library can be changed, see [Library Versions](#) on page 143.
- Types can be added to a library, as long as its state is Open, see [Add Types to Libraries Used in Applications](#) on page 147 and [Add Customized Hardware Types to Library](#) on page 150.
- A library can only be deleted if it is not connected to any application, library or controller, or if any type is in use in any project in the system (see [Delete Libraries](#) on page 141).

- A library can be password-protected, see [Library Password Protection](#) on page 146.

Connect Libraries

All libraries have to be present in the Library Structure in Plant Explorer, in order for them to be connected to control projects, other libraries, and applications.

All AC 800M standard control software libraries are added to the Library Structure when the AC 800M Connect is added to the 800xA system, see [Figure 41](#). In Project Explorer, libraries connected to a control project are stored in the Libraries folder, while libraries connected to applications and libraries are stored in the Connected Libraries folder, see [Figure 42](#).

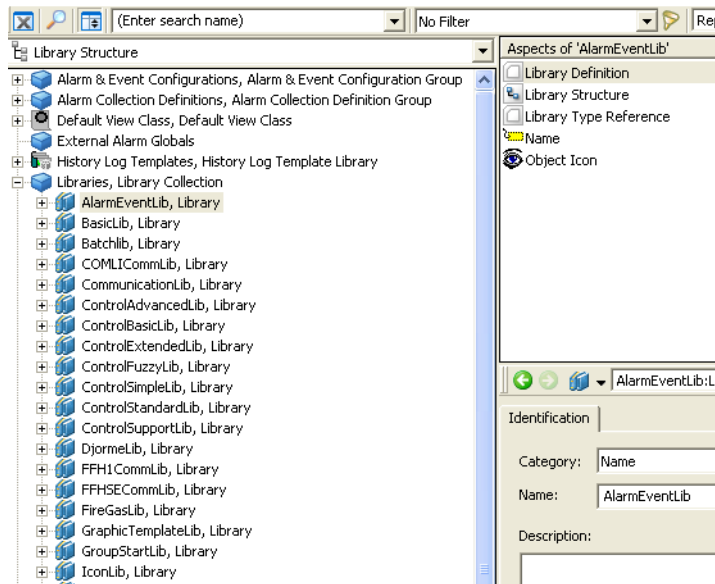


Figure 41. Libraries in Library Structure.

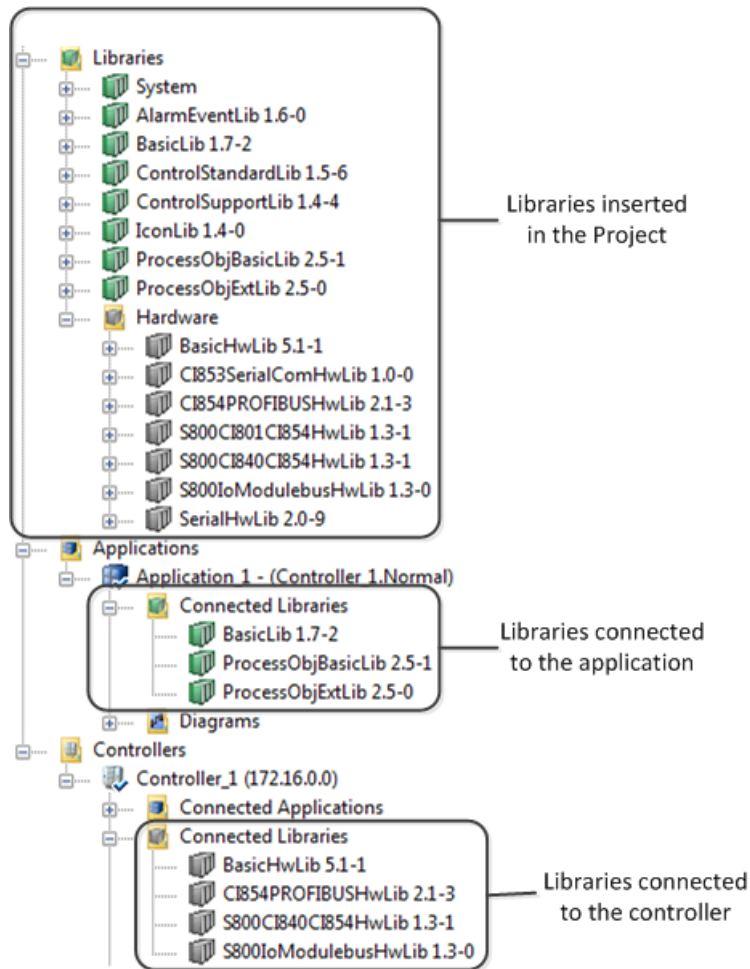


Figure 42. Libraries in Project Explorer

Insert Libraries into Control Projects

A library always has to be inserted into the control project before it can be connected to an application or a controller. To connect a library to a control project:

1. In Project Explorer, expand the Project folder.
2. Select the Libraries/Hardware folder, right-click it and select **Insert Library**.



Libraries can also be inserted in Plant Explorer. Find the project in the Control Structure, select the Project aspect, select the Libraries tab, click **Insert** and select the library from the Select a Library dialog box.

Connect Library to Application, Library or Controller

To connect a library to an application, a library or a controller:

1. In Project Explorer, expand the corresponding Library, Application or Controller folder.
2. Select the corresponding Connected Libraries folder, right-click and select **Connect Library**.



It is also possible to connect a library using drag-and-drop operation. Select the library to be connected, and drag it to the required application, library, or controller folder.

Replace Connected Library

A connected library can be replaced, for example, when the user wants to update to a newer library version. Replacing to a newer version, results in that all instances of a type in the new library will be used instead of the type in the old version.

To replace a connected library:

1. In the corresponding Connected folder, right-click the library and select **Replace Library**.
2. Press the **Yes** button and select a library from the drop-down list in dialog box.
3. Click the **Replace** button to confirm.

Library Usage

The Library Usage function displays the list of places where a library is used, and where it is connected. For ordinary libraries the Library Usage function searches applications and other libraries. For libraries with hardware, it searches controllers.

1. Right-click the library and select Library Usage as in [Figure 43](#). The Library Usage dialog box is displayed with list of applications where the library is connected.

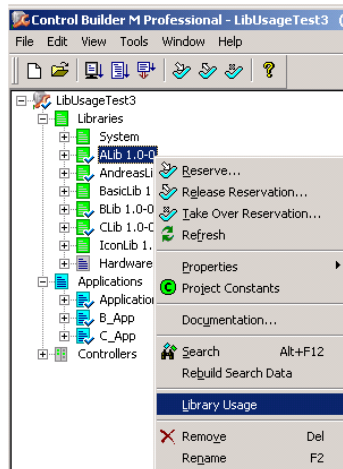


Figure 43. Library Usage

2. Select System to search all projects in Aspect Directory. Click Refresh as shown in Figure 44 to see the library used in several projects.

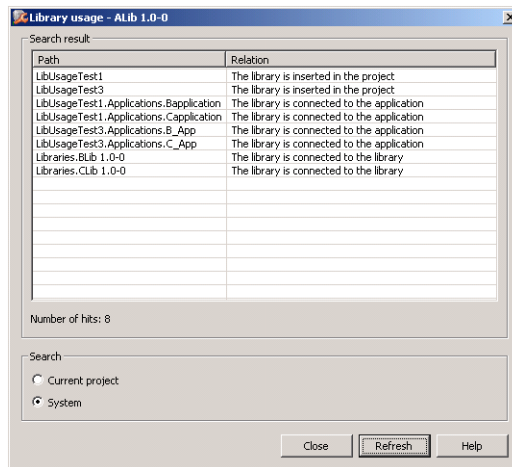


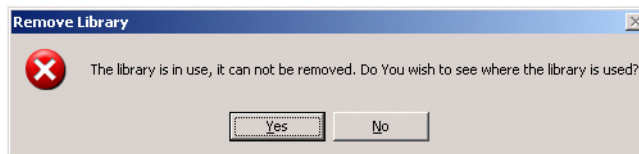
Figure 44. Library Usage dialog box when the System search option is selected

Disconnect Libraries

A library can only be removed if the library and its types are not used within the system.

To remove a library from a control project:

- In the Libraries/Hardware folder, right-click the library and select **Remove**. The library is removed from the control project, but it can be inserted at any time, since it is still present in the Library Structure.
- If the Library is in use the following dialog box displays.



- Click Yes to see the Library Usage dialog box.

Libraries can be disconnected from both applications, libraries and controllers:

- In the corresponding Connected folder, right-click the library and select **Disconnect (Library)**. The library is disconnected, but it can be re-connected at any time, since it is still inserted to the control project.

Delete Libraries

Standard libraries cannot be deleted. Other libraries can be deleted only if they are not connected to any application, library or controller. If you attempt to delete a library with connections to other objects, you will get an error message.

To delete a library from the Library Structure:

1. In the Libraries, Library Collection folder, right-click the library and select **Delete**.

Import/Export Libraries

Libraries can be imported to and exported from an 800xA system. This makes it possible to develop libraries centrally, after which they can be added to other engineering stations at other sites.



For detailed information on how to import/export libraries, see [Import and Export](#) on page 413.

Create Libraries

To create a new library:

1. In Project Explorer, right-click **Libraries** or **Hardware** and select **New Library...** The New Library dialog box is displayed.

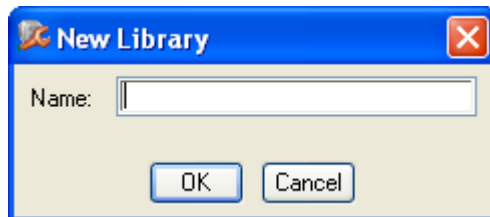


Figure 45. New Library dialog box

2. Enter the name of the new library and click **OK**. The new library is created and inserted into the control project. It is also inserted into the Library Structure in Plant Explorer.



For information on naming conventions for libraries, see *System 800xA Control AC 800M Planning (3BSE043732*)*, and *AC 800M Library Object Style Guide (3BSE042835*)*.

Library States

A library is always in one out of three possible states:

- **Open**
The contents of the library can be changed. This is the normal state for a library when it is under development.
- **Closed**
The contents of the library cannot be changed. However, the state can still be changed back to Open.
- **Released**
The contents of the library cannot be changed. However, in Plant Explorer the state can be changed to Open, but with the Revision index of the version number increased.

To change the library state:

1. In Project Explorer, right-click the library and select **Properties>State**. The State dialog box is displayed.

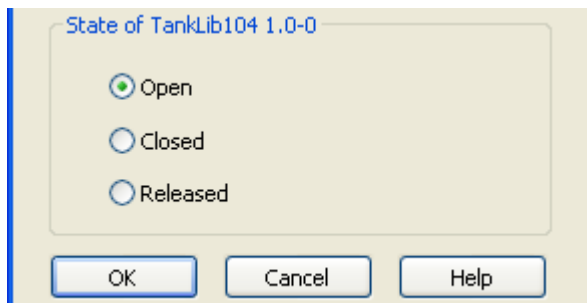


Figure 46. State dialog box

2. Select the desired state and click **OK**. The library state is changed.

The library state can only change:

- From Open to Closed or Released.
- From Closed to Open or Released.

Library Versions

The following rules should be used when creating new versions of a library. The version number syntax is MajorVersion.MinorVersion-Revision (X.Y-Z), for example, 2.0-1.

Table 13. Version handling rules for libraries.

Increase of	Rule	Compatibility with previous versions
Major vers. X	<p>The major version number is increased if the library has types which have changed their behavior, or if it is dependant on a new system version, for example, using new system functions.</p> <p>The major version number is also increased if a connected library has increased its major version number, and the new functionality of this new library version is needed.</p> <p>The maximum limit for the major version number of a library or a hardware library is 32767.</p>	The library is system or application incompatible.

Table 13. Version handling rules for libraries.

Increase of	Rule	Compatibility with previous versions
Minor vers. Y	<p>The minor version number is increased if new types have been added to a library, or an already existing type has increased functionality.</p> <p>The minor version number is also increased if a connected library has increased its minor version number, and the new functionality, which is the reason for the change, is needed.</p>	<p>The library is compatible.</p> <p>The increased minor version number reflects extended, modified, or added functionality.</p>
Rev. Z	<p>The revision index is increased when only bug fixes have been done or when library state is changed from Released to Open.</p> <p>The revision number is also increased if a connected library has increased its revision number, and this new version is needed.</p>	<p>The library is compatible.</p> <p>Functions may now have changed their behavior, since they are working as intended. This may affect the application behavior.</p>

The library version can be changed in two ways:

- Change Library Version** (Project Explorer)

This operation only works on libraries with state Open. This operation does not create a new copy of the library. It simply updates the version number (that is, it changes the version label of the library). The new version replaces the old and all connections to other objects are intact.
- Create New Library Version** (Plant Explorer)

This operation creates a new version of the library. This new version exists in parallel with the old version. All connections to control projects, applications and other libraries are preserved in the old version, but the new version does not preserve any connections.

The two versions cannot be connected the same application or library, but they can be inserted into the same control project.

Change Library Version

The library version can only be changed for libraries with state Open. To change the library version:

1. In Project Explorer, right-click the library and select **Properties>Version**. The Version dialog box is displayed.

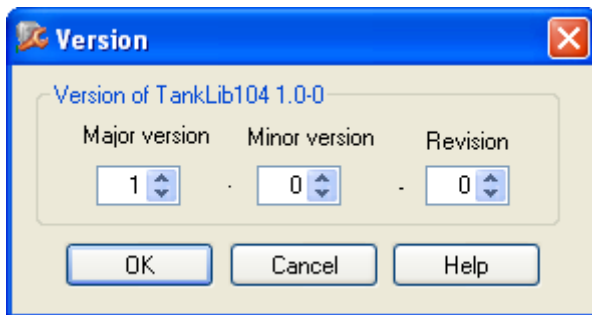


Figure 47. Version dialog box

2. Set the new version number, according to the version handling rules, see [Table 13](#) on page 143.
3. Click **OK**. The version number of the library changes.

Create New Library Version

To create a new library version:

1. In the Library Structure in Plant Explorer, expand the Libraries, Library Collection folder.



A new version can only be created if the library state is Released. If you try to create a new version of a library with state Closed or Open, you will get an error message.

2. Click the library and select **Library Version Definition** aspect. The Aspect preview pane opens.
3. Click **New Version** button. A 'New Version' dialog box opens ([Figure 48](#)).

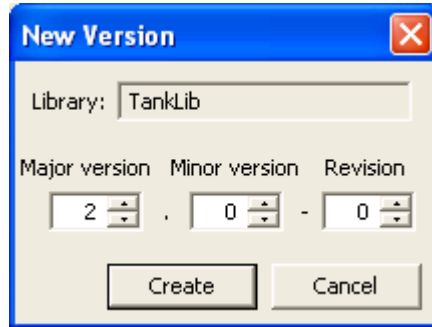


Figure 48. New Version dialog box

4. Enter a new version number according to the version handling rules, see [Table 13](#) on page 143.
5. Click **Create**. A new version of the library is created.



The new library version are not used anywhere by default, thus you must connect/replace the library yourself.

Advanced Library Version Handling in Applications

For a detailed discussion on how to work with library versions (libraries that have types to be used in applications), see the *System 800xA Control AC 800M Binary and Analog Handling (3BSE035981*)*.

Library Password Protection

To password protect the libraries:

1. Right-click the library and select **Properties > Protection**. The Protection Properties dialog opens.
2. Click **Set Password**. The Password dialog opens, see [Figure 49](#).

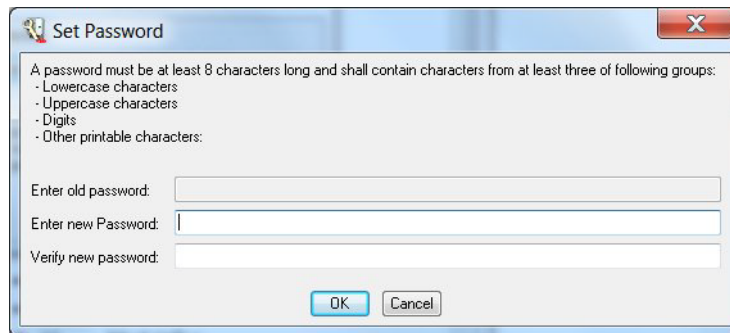


Figure 49. Password dialog box

3. Enter the new password and confirm it in the Verify new password field.



If the library is already password protected, you have to enter the old password before entering a new one. A password must be at least 8 characters long and should contain characters from three of the following groups:

- Lowercase characters
- Uppercase characters
- Digits
- Other printable characters

4. Click **OK**. The library can now not be changed without entering the password.

Add Types to Libraries Used in Applications

Types can only be added if the library state is Open.

Follow the steps below to add the following functions in a library:

1. In Project Explorer, expand the corresponding library folder.

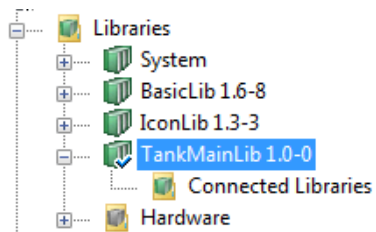


Figure 50. Library with sub folders

2. To the library (see [Figure 50](#)), add the following:
 - a. To connect another library to library, right-click the Connected Libraries folder and select **Connect Library**.

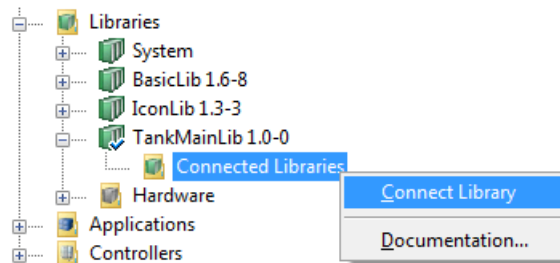


Figure 51. Connecting a Library

- b. To add project constants to library, right-click the library folder and select **Project Constants**.

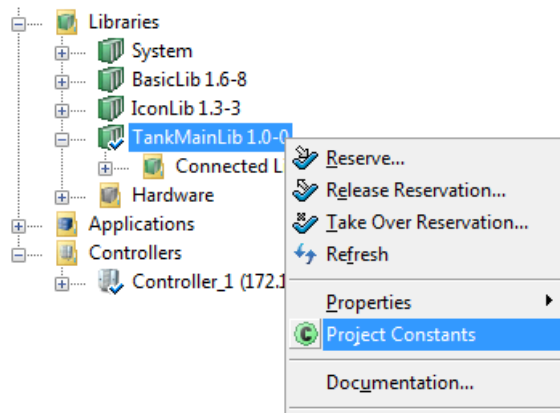


Figure 52. Adding Project Constants

- c. To add a type to the library, right-click the folder corresponding to the type you want to add and select the command for creating a new type.

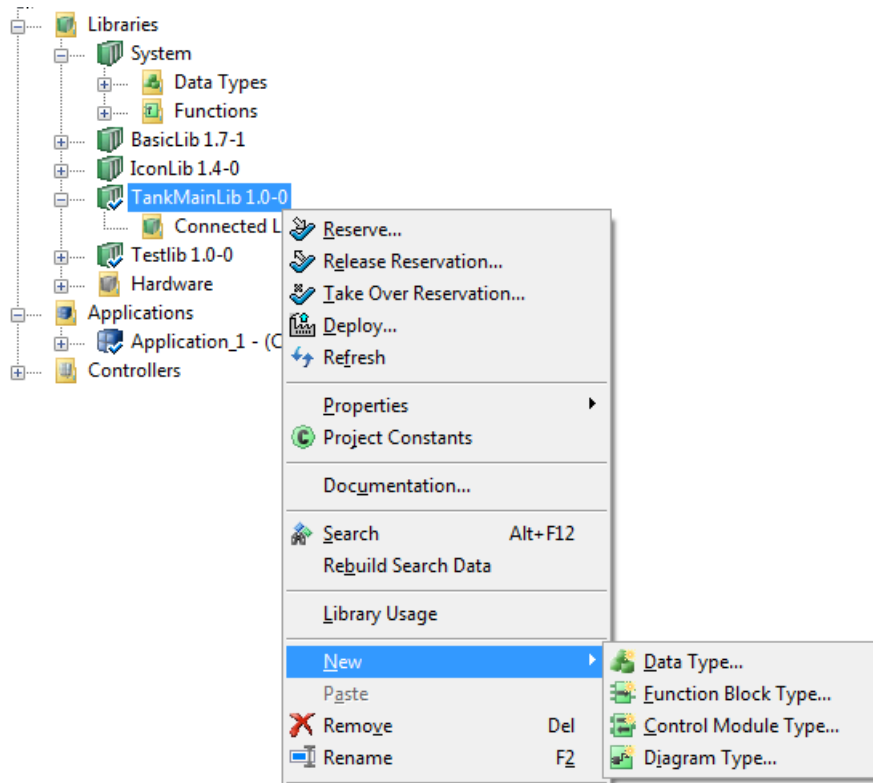


Figure 53. Adding a type

Add Customized Hardware Types to Library

Customized hardware types can only be added to the library if the library state is Open. To add a customized hardware type to a library:

1. In Project Explorer, expand Libraries > Hardware.
2. Right-click Hardware types folder under your chosen library, and select **Insert/Replace Hardware Type(s)**.

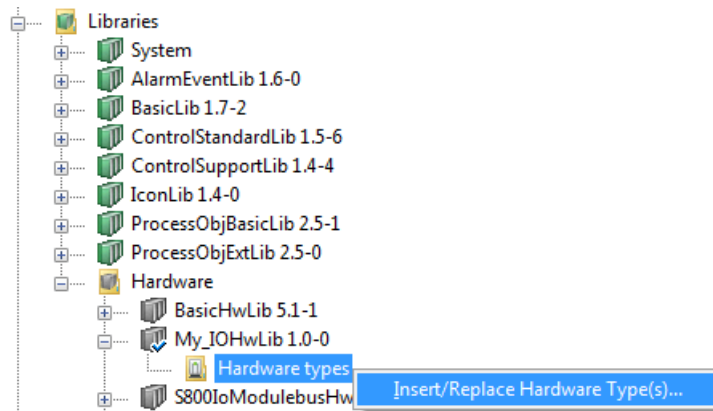


Figure 54. Inserting hardware types in library

3. Browse and select the device capability description file (for example a *.gsd file) you want to add as hardware and click **Open**. (See also [Supported Device Capability Description Files](#) on page 152).
4. The Device Import Wizard starts. Follow the instructions in the wizard.

The usual way to distribute and share customized hardware types is to Export and Import the complete library (with the customized hardware type(s)), in Plant Explorer. In exceptional cases, it is possible to insert individual external customized hardware types to a user-defined library, for example, a hardware type of a *.gsd file that have been converted and used in an earlier version of Control Builder.

In this case, right-click the Hardware types folder under your chosen library and select **Insert/Replace Hardware Type(s)** and browse to the hardware type (*.hwd file) to be inserted. With **Insert/Replace Hardware Type(s)** it is also possible to replace same hardware type.



If a hardware definition file (*.hwd) is re-imported with changed parameters, Control Builder must be restarted so that the changes take effect.



If changes are made to existing *.hwd files, a new GUID is created for them to coexist after the re-import.

Device Import Wizard

You use this wizard to import a device capability description file. The wizard will convert this file to a hardware type and insert the type into a user-defined library. The appearance of some wizard dialog boxes will be different depending on the file type to import.



Always complete the wizard, even if you are not finished. Then, you can re-import the file and continue where you left off.



When a wizard dialog box is displayed, relevant information is read from the device capability description file. If it is large this may take a while, and a progress bar will be shown.

- You can import a new device capability description file, as described above ([Add Customized Hardware Types to Library](#) on page 150).
- You can change conversion settings for a previous import, as described in [Wizard](#) on page 155.
- When you receive an updated device capability description file, you may want to replace the previous import. Import the new file the same way as the old one, as described above.



For more information on the Device Import Wizard, refer to the online help.

Supported Device Capability Description Files

You can only import supported device capability description files. The following files are supported:

- PROFIBUS GSD files
- PROFINET IO GSD files
- Ethernet/IP and DeviceNet EDS files

For PROFIBUS GSD-files, *.gs? is the standard file extension. However, a file can also have a different extension that specifies its language, for example, *.gse (English) or *.gsg (German).

For PROFINET IO GSD files, *.xml is the standard file extension. PNIO uses GSDML, an XML based markup language to describe the characteristics of the PNIO devices.

For Ethernet/IP and DeviceNet, *.eds is the standard file extension. The wizard will convert the EDS file to a hardware definition file (HWD File) and insert it as a hardware type into the user-defined library.



You can only import PROFIBUS GSD-files with hardware types for CI854, and not for CI851. (However, when you upgrade a previous system offering, any included hardware types for CI851 will be upgraded as well.)



For more information on using Device Import Wizard for importing gsd, xml and eds files into the Control Builder, refer to:

- *AC 800M, PROFIBUS DP, Configuration (3BDS009030*)*.
- *AC 800M, ProfiNet IO, Configuration (3BDS021515*)*.
- *AC 800M, EtherNet/IP DeviceNet, Configuration (9ARD000014*)*

Additional Files for Libraries with Hardware

There are a number of files associated with libraries for hardware and hardware types. For standard system libraries, it is not possible to perform any operation on these type of files. For a user-defined library there are some files that can be managed.

The file types, described below, are associated with the hardware definition and cannot be changed or replaced.

File Types Associated with Hardware Types

To display the Additional Files dialog box for a hardware type:

1. In Project Explorer, expand the library with the hardware type under Libraries > Hardware.
2. Under Hardware types for the library, right-click the hardware type and select **Files**.

The only file type (in a user-defined library) that the user can perform any operations on is the Help File. See [Help File](#) on page 155.

The file types, listed in [Table 14](#), are associated with the hardware type and cannot be modified by the user.

Table 14. File Types Associated with Hardware Types

File Type	Description
Firmware File	Firmware file for CPU or communication interface unit.
Update File	Update file for firmware; a download support file.
Firmware Idx File	Idx file for firmware, used when analyzing a crash dump.
Protocol Handler Control Builder File	Protocol handler used by Control Builder.
Protocol Handler Controller File	Protocol handler used by controller.
Protocol Handler Idx File	Idx file for controller protocol handler, used when analyzing a crash dump.

File Types Associated with Libraries



It is only possible to manage Additional files for a user-defined library.

To display the Additional Files dialog for a library with hardware types:

1. In Project Explorer, browse Libraries > Hardware.
2. Right-click the library and select **Properties > Files**.

The file types, listed in [Table 15](#), are associated with the library.

Table 15. File Types Associated with Libraries

File Type	Description
Help File	A help file (of *.chm type) can be added, replaced, deleted or extracted, See Help File on page 155
Import File	Import file is a device capability description file (for example a *.gsd file) that has been added with the Device Import Wizard. This type of file can be deleted (Delete button), or extracted (Extract button) to a file on disk. By pressing the Wizard button it is also possible to change the previous done settings. See Wizard .

Wizard

Settings for a previously added device capability description file can be changed.

1. In Additional Files for a library, select the row with the device capability description file (Import File) and press the **Wizard** button.
2. In the displayed Device Import Wizard, define the new conversion settings.

Help File

A help file (of *.chm type) can be added, replaced, deleted or extracted for a customized hardware type, as well as for a user-defined library.

Adding a help file to a customized hardware type or a user-defined library provides access to the associated help file when you press F1 on the user-defined library or on the customized hardware type, in Project Explorer. For further information about requirements on customized online help, see the *System 800xA Control AC 800M Binary and Analog Handling (3BSE035981*)*.

To add a help file to a user-defined library or to a customized hardware type:

1. In Additional Files dialog box, select the **Help File** row and press the **Add** button.

Browse to the help file (of *.chm type) and click **Open**.

Replace and Delete

A help file that has been added can be replaced and deleted by selecting the row with the help file and pressing **Replace** and **Delete** button respectively. It is also possible to delete a device capability file (Import File) for a user-defined library.

Extract and Save a Copy of a File

A help file can be extracted and saved on disk by selecting the row with the help file and press the **Extract** button (to the right of the grid). Browse to a place on disk and save a copy of the file by pressing **Save** button.

In some exceptional cases there is a need to extract an individual customized hardware type to a hardware definition file (*.hwd file). In this case, press the **Extract** button under *Hwd File*.

Properties on Hardware Types

In Additional Files for a customized hardware type, it is possible to set a version information text of maximum 18 character to the help file, by pressing the **Properties** button.

Delete Hardware Types

A hardware type in a library can be removed.



It is not possible to remove a hardware type from a library, if it is used in a hardware configuration, in any project of the system (aspect directory).

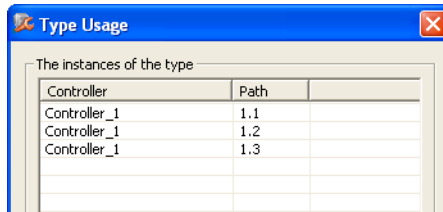
1. In Project Explorer, expand the library with the hardware type under Libraries > Hardware.
2. Under Hardware types for the library, right-click the hardware type and select **Remove**.

Type Usage for Hardware Types

It is possible to display a list of which controller(s) that use(s) the hardware type together with hardware tree position numbers.

1. In Project Explorer, expand the library with the hardware type under Libraries > Hardware.

2. Under Hardware types for the library, right-click the hardware type and select **Type Usage**.



The screenshot shows a dialog box titled "Type Usage" with a close button in the top right corner. Below the title bar, the text "The instances of the type" is displayed above a table. The table has two columns: "Controller" and "Path". There are three rows of data, each showing "Controller_1" in the "Controller" column and a path (1.1, 1.2, 1.3) in the "Path" column. There are also three empty rows below the data.

Controller	Path
Controller_1	1.1
Controller_1	1.2
Controller_1	1.3

Figure 55. Type Usage for a selected hardware type.

Hide and Protect Control Module Types, Function Block Types, Diagram Types, and Data Types

When you create libraries with self-defined control module types, function block types, diagram types, and data types, Control Builder provides you with two protection features (attributes). These two attributes are called *Hidden* and *Protected*, and can only be set from Project Explorer.

Hidden

Setting the Hidden attribute will completely hide your code from other users. To hide the code makes it easier to improve your type as often as you like. This is a common situation when developing types that will be re-used over and over again in different library solutions.



All types with the Hidden attribute disappear from their normal position in the Object Type Structure, and can only be located in the Internal Types folder, as a Hidden aspect.

Protected

Setting your type to Protected will protect the internal type structure from being seen. This means that only the type itself will be visible, and thus your type definition will be protected from external exposure, as well as any attempt to duplicate it. This is extra valuable when you create a type solution for re-use engineering.

When a protected type is opened in an editor, only the parameter declaration is visible (read-only). Other declarations like variables and function blocks are not visible. Only code block tab is shown as a blank page, that is, the IEC 61131-3 logic is protected. The complete type structure will still be protected from external exposure.

The attribute available on protected control modules and function blocks types "Sub Objects visible in PPA" makes formal instances in the protected type visible in Plant

Explorer if they are configured as aspect objects. It does not make the subobjects visible in Control Builder.



The Hidden and Protected attribute can also be used for structured data types.

Override

After you have protected your types, you can always override the hidden and protected attribute temporarily, while you work on improvements. The override protection property can only be set in Project Explorer.

For self-made libraries with password protection, you must enter the password before you make an override, see [Library Password Protection](#) on page 146



The protection cannot be overridden for Control Builder standard libraries. They cannot be updated or changed by the user.



Setting an override on a library for corresponding hidden and/or protected types will only have impact in Project Explorer. In Plant Explorer, hidden and/or protected types will remain hidden and/or protected.

Protect a Self-Defined Type

To protect a self-defined type:

1. In Project Explorer, right-click the type and select **Properties > Protection and Scope**. A Protection and Scope window opens.

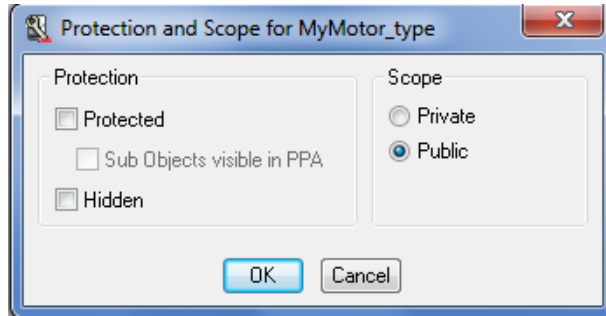


Figure 56. Protection and scope

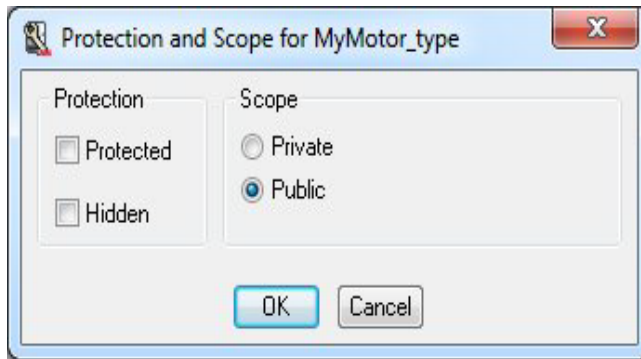


Figure 57. Protection and scope

2. Check the desired protection radio button(s) and click **OK**.

Override Protection Attributes

To override protection for a library or application:

1. In Project Explorer, right-click the library (or application) and select **Properties > Protection**. A Protection Properties window opens.

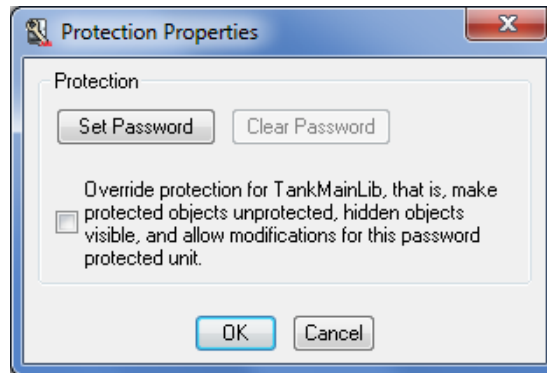


Figure 58. Protection properties

2. Check the Override check box (see figure above) and click **OK**. The Override feature will have impact in Project Explorer only.

Protect MySupervision Type Example

The following example will show the impact that the Hidden and Protected attributes may have on a self-defined type called MySupervision_type, which is part of the library MyTankLib.

A Simplified Library Solution

The library MyTankLib contains three different types, MyMotor_type, MySupervision_type and MyTank_type. The Motor10 object is located inside MyTank_type, see [Figure 59](#). As you can see, Motor10 has inherited its definition

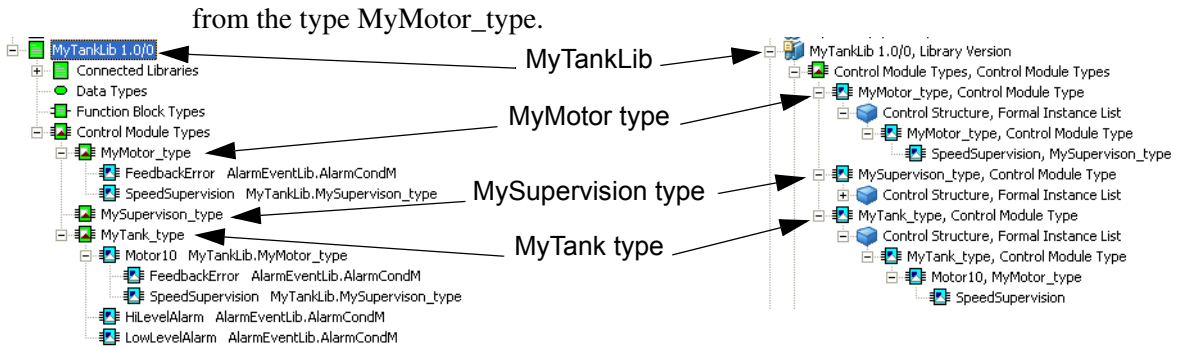


Figure 59. A Library structure before any protection attributes have been set. (Left) Project Explorer tree. (Right) Object Type Structure in Plant Explorer.

Protect MySupervision Using the Hidden Attribute

MyMotor_type contains a SpeedSupervision object (and a Feedback error object). The SpeedSupervision object is of the type MySupervision_type. Both MyMotor_type and the MyTank_type therefore depend on MySupervision_type. To hide the code inside the MySupervision_type, we must set the attribute Hidden on MySupervision_type, see Figure 60.

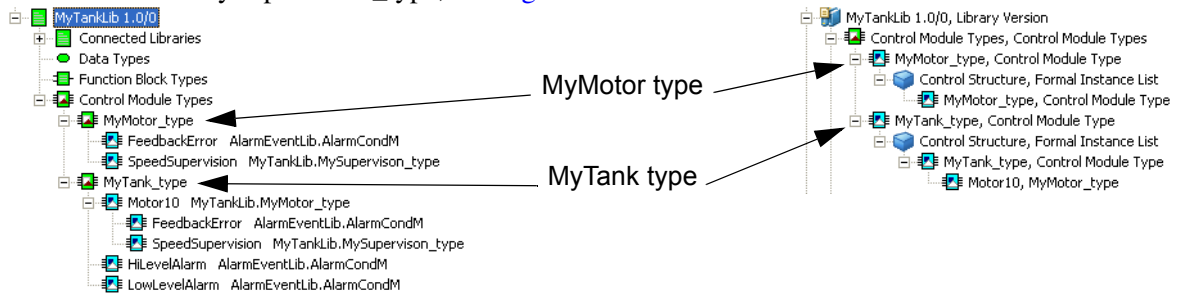


Figure 60. MySupervision_type is not shown in Plant Explorer after setting the hidden attribute. (Left) Project Explorer with SpeedSupervision still visible. (Right) Object Type Structure where both MySupervision_type and SpeedSupervision are hidden.

After Hidden is set on the Supervision type, it disappears from both the Project Explorer and the Plant Explorer. However, MySupervision type can still be traced via calls from the SpeedSupervision object inside the motor type to our hidden

supervision type in Project Explorer, see [Figure 61](#).

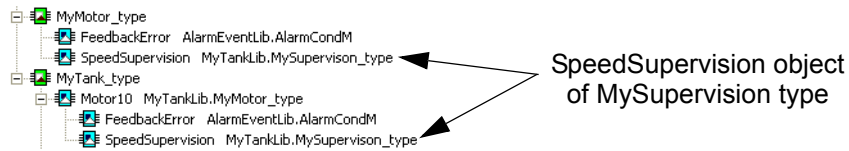


Figure 61. The hidden MySupervision type can still be traced via the SpeedSupervision object in both the motor type and the motor object.

Setting the Protected Attribute for MyMotor_type

If we do not like to expose SpeedSupervision, why not hide the motor type as well? The major reason is that it would be impractical to set hidden on the motor type, just to conceal the function calls from SpeedSupervision (expose the existence of MySupervision_type).

Besides, SpeedSupervision would still be visible in the motor object (Motor10, etc.) inside the tank type, see [Figure 62](#).

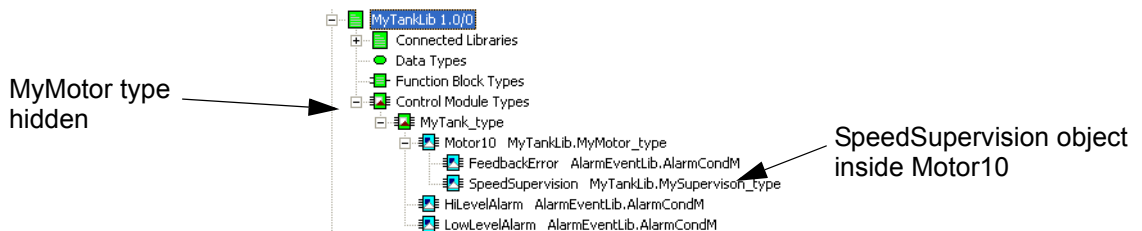


Figure 62. The Hidden attribute on MyMotor type would still allow showing objects (children) of the MySupervision type in any new motor object.

Furthermore, we must be able to select the motor type every time we create a new motor object.

Therefore, for re-usability reasons, we cannot hide the motor type like we did with the supervision type, but, we can set the Protected attribute, since a protected type will still be visible in Project Explorer, while the type definition is hidden according to [Figure 63](#).

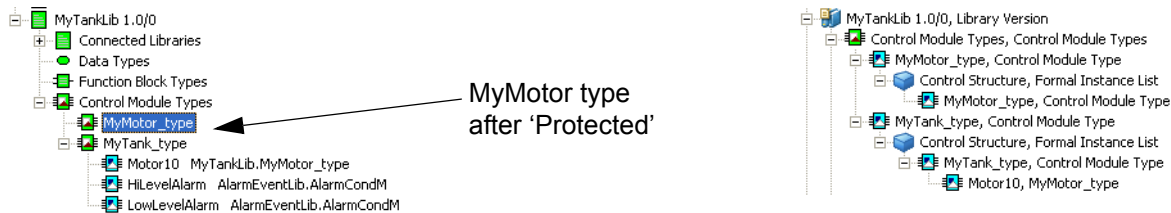


Figure 63. Protected attribute on MyMotortype, which will hide the type definition.

In this case, a protected motor type will still let the user create new motor objects of the type MyMotor_type in other libraries, like the one in Figure 64, but without knowing about the background calls from SpeedSupervision.

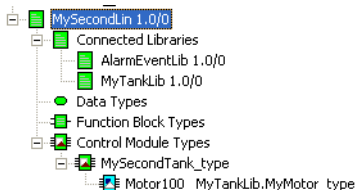


Figure 64. Motor100 object of the MyMotor_type, re-used in another library with MyMotor_type protected and MySupervision_type hidden.



For more information about control module types and function block types, see *System 800xA Control AC 800M Planning (3BSE043732*)*.

Task Control

A *task* is defined as an execution control element that is capable of starting, on a periodic basis, the execution of a set of POU's (Programs, Function blocks, functions etc.).

The Control Builder setup three tasks (Fast, Normal and Slow) by default, provided that an AC 800M Control Project template has been selected. The tasks are connected to their respective diagrams (one task per diagram). The tasks serve as 'work schedulers' for the diagrams and contain settings for interval time and priority. However, setting interval time and priority is not enough; you must also tune your tasks.



To learn how to tune tasks, see *System 800xA Control AC 800M Planning (3BSE043732*)* manual.

If a diagram does not have a task connected, it will run by the task connected to the corresponding Application.

You may create and connect several tasks to a controller, but experience show that more than five tasks in each controller makes it difficult to overview.

The Control Builder provides a Task Analysis tool that predicts the execution of an application by the controller before loading it onto the controller. See [Task Analysis](#) on page 187 for more information.

Task Connections

A task can be connected to a program, a function block, a control module or a single control module, a diagram, and several tasks may execute in the same controller. An application can also be connected to a task, and all POU's in an application execute in this task, unless otherwise specified. A task can only execute POU's in one application. Hence, POU's from different applications can not be connected to the same task.

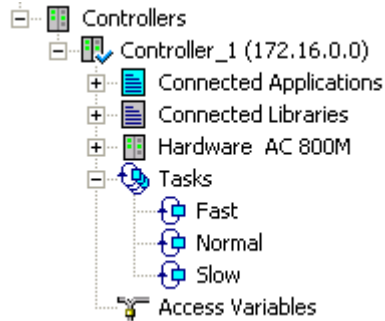


Do not re-connect tasks to applications unless it is necessary, as this might disrupt the task execution during re-configuration. Else change the parameters of the connected task (to fit the needs). A SIL3 task re-connection might lead to a shut down of the controller.

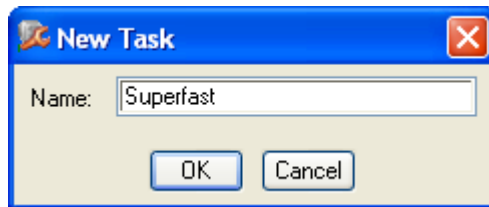
Create a New Task

To create and configure a new task:

1. Expand the Hardware tree, until you find **Tasks**.



2. Right-click **Tasks** and select **New Task**. A 'New Task' window opens.
3. Name the task.



4. Click **OK**.

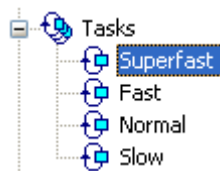


Figure 65. A new task has been created.

After the task has been created, it is time to configure the task with new properties.

5. Right-click the new task (Superfast) and select **Properties**. A 'Task Properties' window opens.

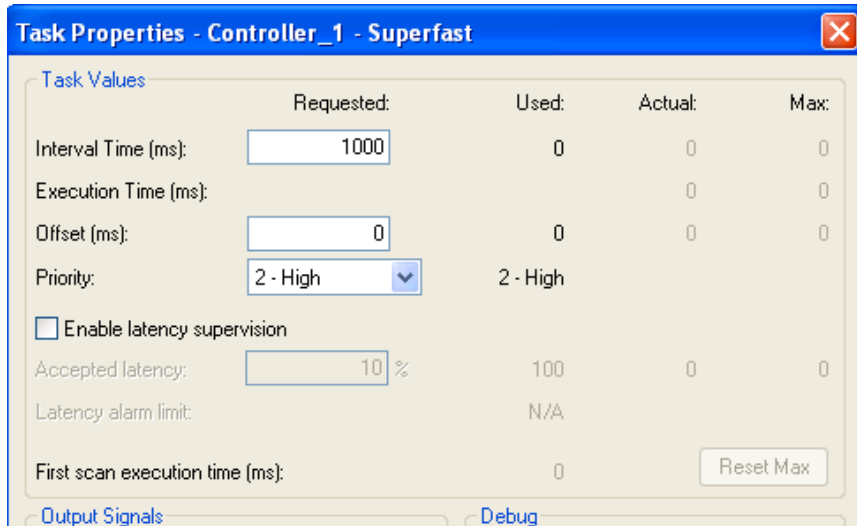


Figure 66. A Task Properties window for configuring a task.

6. Change the interval time to 40 ms and Priority to 1-Highest. Click **Apply** followed by **Close**.
7. Right-click Tasks and select **Editor** to view the new task. A ‘Task Overview’ window opens.

	Name	Priority	Interval Time	Actual Interval Time	Max Interval Time	Actual Execution Time	Max Execution Time	Offset	Actual Offset	Max Offset	Accepted Latency	Actual Latency	Max Latency	Latency Alarm Limit	First Scan Execution Time
1	Superfast	1 - Highest	40	0	0	0	0	0	0	0	N/A	0	0	N/A	0
2	Fast	2 - High	50	0	0	0	0	0	0	0	N/A	0	0	N/A	0
3	Normal	3 - Normal	250	0	0	0	0	15	0	0	N/A	0	0	N/A	0
4	Slow	4 - Low	1000	0	0	0	0	25	0	0	N/A	0	0	N/A	0

The Task Overview window lists all the tasks with each property settings. To change the settings for a certain task:

8. Select a task in the Task Overview window and open **Tools > Task Properties**.



Right-click a task directly in the hardware tree and select **Properties** to open the Task Properties window directly.

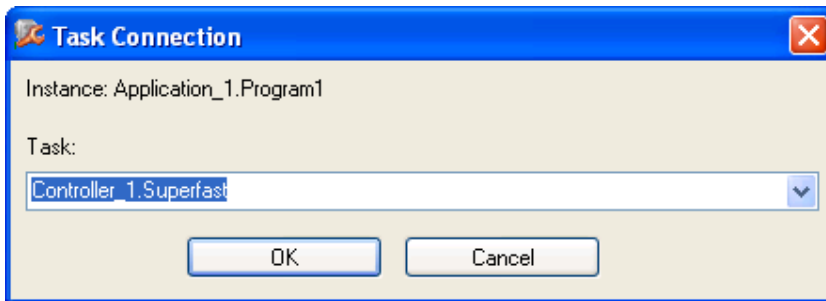


Select **Tools > Reset Max**, to reset all tasks that appear in the editor.

Connect a Task to a Program

To connect the task SuperFast to Program1:

1. Right-click **Program1** and select **Properties > Task Connection**. A ‘Task Connection’ dialog box opens.



2. Select a task from the drop-down menu (here SuperFast) and click **OK**.

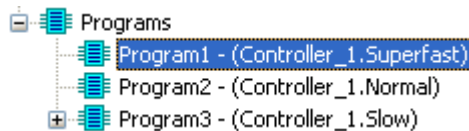


Figure 67. Program1 has changed task to Superfast.

Function Blocks with Different Task Connections

You can connect function blocks inside a program to a task different from the one connected to the program, (right-click on the function block and select ‘Task Connection’).

However, variables inside the function block that pass values to and from the function block are controlled by the program task. The code in the function block will run according to its task, but the parameters will be updated according to the program task. This means, in practice, that the function block in a program can only run at a slower, or a least at the same, speed as the program. However, if you use

external variables or connect I/O directly to the function block, there will be a direct reference, independent of the task cyclicity of the function block.

To set-up specific time intervals and task priority different from the task connected to the application whilst for example, designing libraries, can be done by declaring and using global variables, or by using parameters.



For more information, see [Control the Execution of Individual Objects](#) on page 113.

Task Execution



This sub-section describes priority, interval time and offset for task execution. The next sub-section, [Overrun and Latency](#) on page 178, describes handling of too long task executions, delays, and load balancing etc.

There are four important task parameters that can be set to optimize program execution:

- *Priority*, which sets the execution order for tasks, see sub section Priority below.
- *Interval time*, sets the task intervals during the program is executed, see sub section [Interval Time](#) on page 172.
- *Offset*, a parameter that helps you to avoid unexpected delays in execution when tasks are scheduled to execute at the same time. See sub section [Offset](#) on page 173.
- *Execution time*, for best real time behavior and communication performance, avoid extensive continuous execution. See [Execution Time](#) on page 178 and also [Communication Considerations](#) on page 176.

All POU's connected to a task execute with the same *priority*, *interval time*, *offset*, and *execution time*.

Task Priority

There are six levels of priority: *Time Critical*, *Highest*, *High*, *Normal*, *Low*, and *Lowest*, numbered from 0 to 5. The tasks are executed according to their priority, where the time-critical task has the highest priority. A task with higher priority may interrupt any task with lower priority, but a task cannot interrupt another task with

the same priority. There can only be one time-critical task. Such a task may interrupt the execution at any point, while other tasks may only interrupt execution at defined points.

An ordinary (non-time-critical) task can be interrupted:

- at the start of any code block,
- at backward jumps, for example for, while, repeat statements.

A time-critical task has special properties.

- The task is not driven by the same scheduler as the rest of the tasks. Instead, the task is driven from the system's real-time clock (hence the high precision).
- The tasks have high precision in execution time. The resolution is 1 ms.
- A change to/from time-critical priority in Online mode is not possible.
- A change to/from time-critical priority in Offline mode requires re-compilation of the application.

Consider the following points, when using the time-critical priority.

- Only one time-critical task per controller is allowed.
- The execution time for a time-critical task (priority 0) must not exceed 100ms. This restraint prevents the task from blocking other functions, for example communication.
- All functions cannot be called from the program connected to the task. You cannot set time-critical priority if the code contains invalid instructions (this is checked during compilation). The time-critical task interrupts execution at any time, which means that execution might be interrupted mid-statement.
- If a power failure occurs while the time-critical task is running, the execution of the current code block is completed (assuming that it can be completed within 1 ms). For a warm start to be possible, no code block in the time-critical task may take more than 1 ms to execute.



Task priorities 1–5 can be set by using the firmware function *SetPriority*. This function is located in the System folder.

Consider the following points, when using task priority in HI controller:

- In HI controller VMT has the highest possible task priority. SIL3, SIL2 and non-SIL can not share the same priority and have the priority in order listed.

- Only one task can be connected to a SIL3 application. If more than one task is connected, compilation error is generated. To download remove all tasks except SIL3 task.
- The SIL3 tasks must have higher priority than non-SIL and SIL1-2 tasks in the controller. If not compilation error is generated. Decrease the priority of the non-SIL and SIL1-2 tasks or increase the priority of the SIL3 task to enable downloading.
- It is not recommended to have a task with the same or higher priority than the VMT task, regardless of SIL level. If the VMT task is not the only task with the highest priority, a compilation warning is generated. The user should decrease the priority of any task (SIL or non-SIL) which has the same, or higher priority than the VMT task.
- Firmware functions that tries to manipulate task parameters from 1131 code does not work for SIL tasks that is SetPriority and SetIntervalTime.
- Higher prioritized tasks cannot interrupt a lower prioritized task during IAC fast data copy in/out. If the number of Communication Variables in lower prioritized SIL task exceeds the limits defined in [Communication Variable Limits Dialog](#), then either perform a proper task tuning such that the high priority of the task is not crucial or increase the accepted latency.

Interval Time

The interval time, during which the program is executed, is set in the Task Properties dialog box. Default values are 50 ms (Fast), 250 ms (Normal) and 1000 ms (Slow). You can change these values at any time. For a time-critical task, the interval time can be as short as 1 ms. The interval time of tasks of priority 1–5 cannot be less than 10 ms. The resolution is 1 ms.



If two tasks have the same priority, and they both wait for execution, the task with the shortest interval time will be executed first.



All task intervals must be multiples of each other. The shortest interval is the "time base".

Execution Example

Figure 68 shows two tasks executing in the same system. Task 1 and task 2 have interval times of 30 and 200 ms, and execution times of 10 and 50 ms, respectively.

When the tasks have been assigned the same priority, the execution start time of task 1 is very much delayed. It also drops one execution.

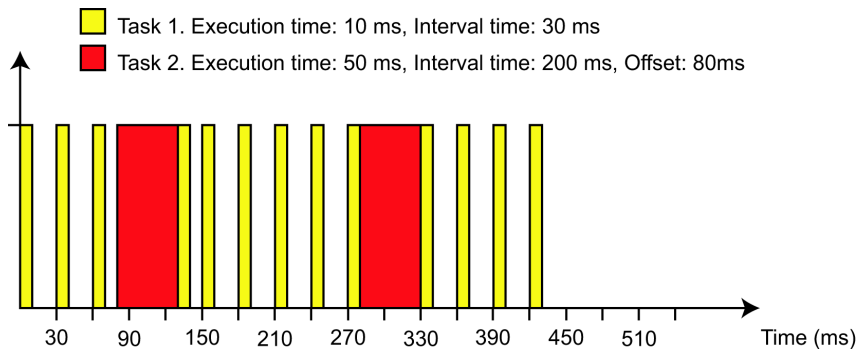


Figure 68. Execution of two tasks with the same priority.

In Figure 69, task 1 has higher priority than task 2, and interrupts the execution of task 2. Hence task 1 is not delayed much by task 2.

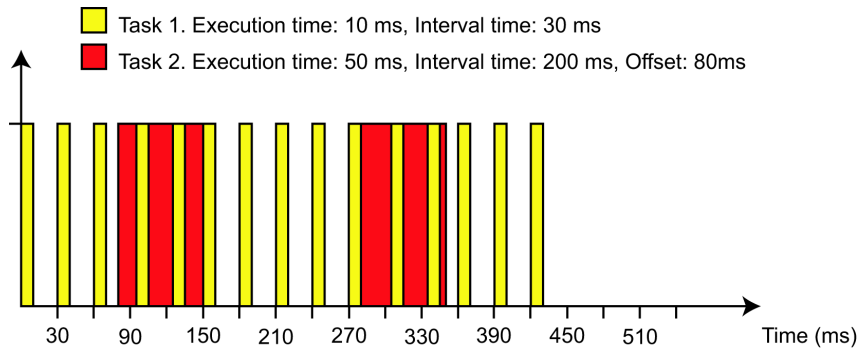


Figure 69. Execution of two tasks with different priorities.

Offset



The compiler will detect inappropriate offset settings.

The offset of each task must be equal or greater than the sum of the execution times of all higher-priority tasks.

If your tasks are scheduled to execute at the same time you will receive a warning during download. However, this compiler function is merely calculating theoretical periodic executions, which means that it will not warn you for task collision caused by, for example a too close offset time. Therefore, consider the compiler warning as a first preliminary check provided to you and not as a guarantee that will prevent task collisions.

Two tasks will be scheduled to start execution at the same time if the greatest common divisor of the tasks interval times divides the difference in the tasks offsets.

Turning off Task Collision warnings

You can turn off the task collision warning from the Project Explorer.

1. Right-click the Project item and select **Settings > Compilation Warnings** from the context menu. A Compilation warnings dialog box will open.
2. Click to clear **Task Collisions** check box and then **OK**.

When tasks are scheduled to execute at the same time, the task with the highest priority will be executed first. If tasks have the same priority the task with the shortest interval time will be executed first. Offset is a mechanism that can be used to avoid unexpected delays in execution when tasks are scheduled to execute at the same time.



Do not change task offset for a controller with a running application. This may result in that the task executes one more time than expected.

In [Figure 70](#) and [Figure 71](#), the execution of two tasks with the same priority with interval times of 50 ms and 100 ms is shown. When both tasks have a 0 ms offset ([Figure 70](#)), the execution start time of task 2 is delayed, and the actual interval time for task 2 is influenced by variations in the execution time of task 1.

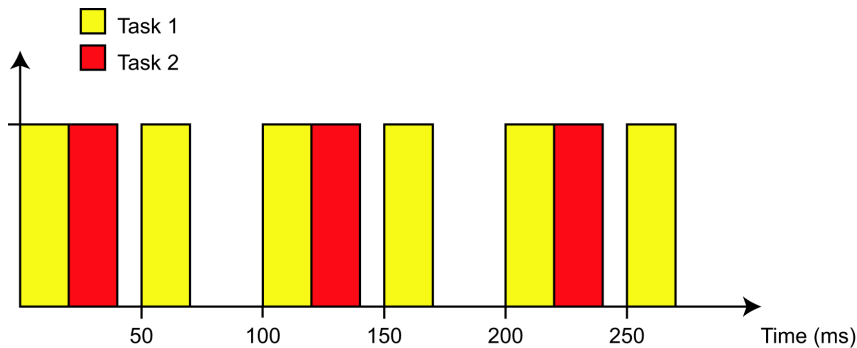


Figure 70. No offset. The two tasks have the same priority, but different interval times (50 and 100 ms).

If task 2 is assigned an offset, as in [Figure 71](#), neither task is delayed, and the actual interval time for task 2 will not be affected by task 1.

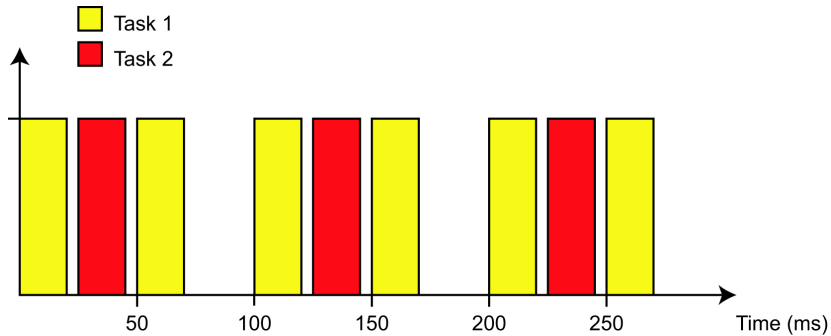


Figure 71. Offset is set on task 2. The two tasks have the same priority, but different interval times (50 and 100 ms) and are thus executed at the requested times.

An application starts to execute by scheduling all tasks in the application to execute at the same time. The task with highest priority is executed first, and if tasks have the same priority, the task with the shortest interval time will be executed first.

Execution Synchronization

When a task has finished execution of the first scan after application start at time t , the start of its next execution is synchronized to time 0 (the time the controller started to execute).

$$t = n * (\text{interval time}) + d, \quad 0 \leq d < \text{interval time}$$

d is the time from the start of the current interval time, to when the task finished execution in the current interval. The synchronization to time zero (0) implies that the start of the next execution will be at the first start point after the current time.

If offset = 0, the task will be scheduled to execute at time $(n + 1) * (\text{interval time})$. However, if the time to the start of the next execution, $(\text{interval time}) - d$, is less than 10 ms, the task will be scheduled to execute a time $(n + 2) * (\text{interval time})$.

If offset > 0, then if offset > d , the start of the next execution will be at a time $n * (\text{interval time}) + \text{offset}$. If offset < d , the start of the next execution will be at a time $(n + 1) * (\text{interval time}) + \text{offset}$. If the time to the start of the next execution is less than 10 ms, the interval time will be added to the start time of the next execution.

The same synchronization of execution time will be performed after a change in interval time or offset.

Time critical task is not synchronized to time zero (0).

Communication Considerations

POU execution has higher priority than other functions, such as communication. These functions are performed in the gaps between the execution of different tasks. If several tasks with long execution times are executed immediately, one after the other, the time gaps are few but long (see Figure 72).

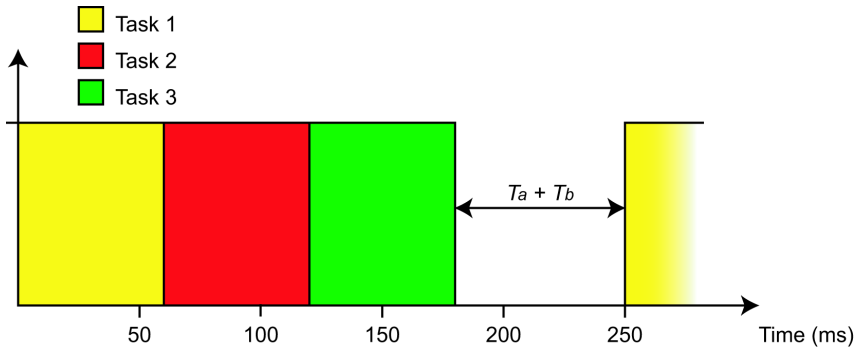


Figure 72. The result of having no offset for three tasks with long execution times. The gap ($T_a + T_b$) is the time available for the execution of other functions, for example communication.

The offset mechanism can be used to make the time gaps more frequent (see Figure 73).

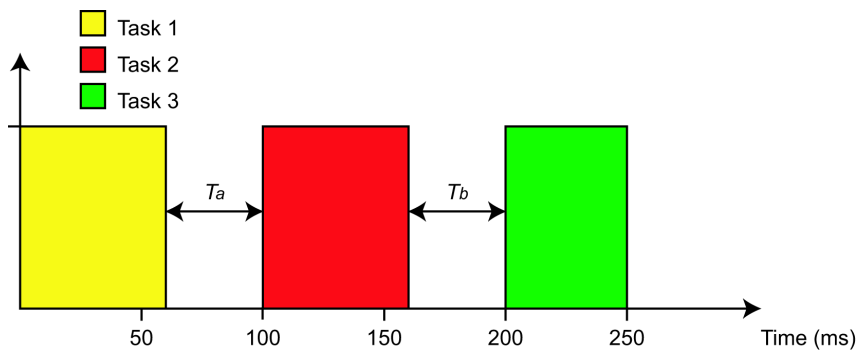


Figure 73. The result of assigning offset to tasks 2 and 3, is that the time available for the execution of other functions occurs more often (T_a).

The same processor handles communication and IEC 61131-3 code. This means that you have to consider how much code you include in each task, when you tune the tasks.

Assume that we have a task running code with an execution time of 500 ms and an interval time of 1000 ms. This means a cyclic load of 50% (load = execution time / interval time). But, this also means that no communication can be performed during the 500 ms execution (since communication has lower priority than the task).

Now, assume that we have divided the code into 4 tasks such that each one corresponds to 125 ms of the execution time. The interval time is still 1000 ms, hence the load is still 50%. But, if we set the offset for the 4 tasks to 0, 250, 500, and 750 ms, the result will be completely different. Now, code will be executed for 125 ms, after which there will be a pause when communication can be performed. Following this, code will be executed for another 125 ms followed by another pause when further communication can be performed. Hence, we still have the same cyclic load, but the possibility for communication has increased considerably.

To conclude, try to tune your tasks using offsets before you change the priority. Actually, the only time you have to change the priority, is when two tasks have so much code that their execution cannot be “contained” within the same time slot, that is, the total execution time exceeds the length of the time slot. It is then necessary to specify which of the two tasks is most important to the system.



More information about task tuning can be found in the *System 800xA Control AC 800M Planning (3BSE043732*)*.

Execution Time

The maximum allowed execution time for time-critical tasks must not exceed 100ms. For load balancing purposes this is also recommended for all tasks in a AC 800M High Integrity controller. In a PA controller, the execution time should not exceed 200 ms. When a task executes for a longer period of time this will disturb/starve other lower prioritized functions in the system such as communication (MMS and others). See [Communication Considerations](#) on page 176.



It is recommended to split applications in several tasks (or even split applications into several applications) such that each task execution time is less than 100 ms and to use task offset configuration to allow for other system functions to execute before the next 1131 task is scheduled.



The maximum allowed execution time does not include the first scan execution time.

Overrun and Latency

Overrun and Latency are two functions for supervising a task. Overrun checks if each task finishes before it is supposed to start the next time, and detects if the task runs for too long. Latency on the other hand, checks that a task starts on time (on each cyclic start), and detects if the task starts too late.

The Overrun function is configured per controller via the Controller Settings dialog box, while the Latency function is configured per task (and SIL classification per task) via the Task Properties dialog box. Both Overrun and the Latency function uses the Error Handler to report any errors.



For High Integrity controllers:

Overrun Supervision is automatically enabled and cannot be switched off. Load balancing is not available in High Integrity controllers.

Latency Supervision is mandatory and therefore automatically enabled for all SIL tasks.

Overrun Supervision

Overrun occurs when the execution of a task takes too long, that is, the task is still executing when the next execution of the task is scheduled to start.

By setting the maximum number of consecutive overruns allowed (missed scans), you can control when a fatal overrun error is considered to have occurred, and consequently configure a controller reaction.

These reaction settings are:

- Nothing,
- Stop Application,
- Reset Controller.

In an AC 800M (non-Hi integrity) controller, load balancing and overrun supervision functions are mutually exclusive, whereas the Load Balancing function is default. Hence, the overrun supervision is turned off. For more information about load balancing and cyclic load, see [Load Balancing](#) on page 184.

Configuring Overrun Supervision

Overrun supervision is set for each controller in the Controller Settings dialog box. To select Overrun Supervision for a controller, follow these steps:

1. Expand the Hardware tree until the controller (for example, Controller_1).
2. Right-click the controller and select **Properties > Controller Settings** from the pop-up menu. A 'Controller Settings' dialog box opens.

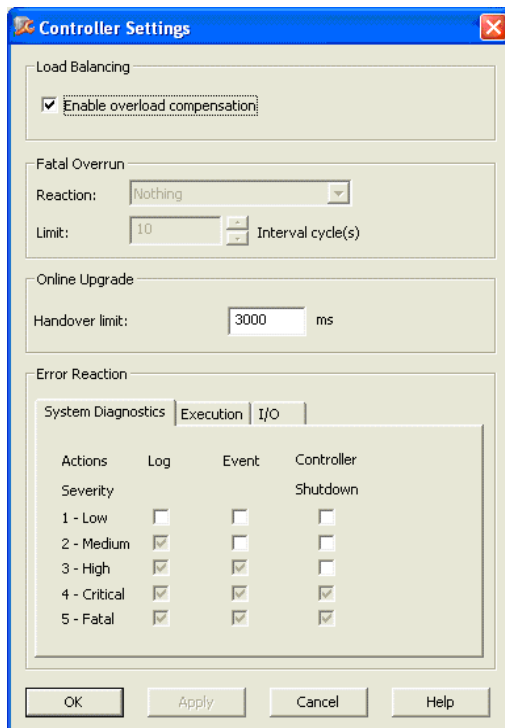


Figure 74. Controller Settings dialog box for a non-High Integrity AC 800M controller.

3. Uncheck Load Balancing, (**Enable overload compensation** check box).
4. Select a reaction for Fatal Overrun from the **Reaction** drop-down menu, (*Reset Controller* or *Stop Application* will activate the Limit field).
5. Enter the number of consecutive overruns allowed in the **Limit** field, (number of consecutive overruns before a fatal overrun is considered to have occurred).
6. Use the tabs under Error Reaction to set-up actions for different error types and severity. (For information on Error Reaction settings, see [Controller Settings in Non-High Integrity Controllers](#) on page 424).
7. Click **OK**.



If overrun errors occur, re-program the faulty task to decrease load.

Latency Supervision

Latency occurs when the execution of a task is delayed, that is, the task starts to execute later than scheduled. The latency function will supervise your tasks (start on time on each cyclic load), and detect if a task starts sooner or later than scheduled.

Latency is activated in the Task Properties dialog box, where you set the acceptable latency in percent (accepted latency in percentage of the interval time). The lowest accepted value for Latency Time is always 10 ms.

Configuring Latency Supervision

Latency supervision is set for each task in the Task Properties dialog box. To select Latency Supervision for a task, follow these steps:

1. Expand the Hardware tree, until you find **Tasks**.

- Right-click a task and select **Properties** from the pop-up menu. A ‘Task Properties’ dialog box opens.

Enable Latency supervision check box

Task Values	Requested:	Used:	Actual:	Max:
Interval Time (ms):	250	0	0	0
Execution Time (ms):			0	0
Offset (ms):	15	0	0	0
Priority:	3 - Normal	3 - Normal		
Accepted latency:	10 %	25	0	0
Latency alarm limit:		N/A		
First scan execution time (ms):		0		Reset Max


- Select Latency, (check **Enable latency supervision** check box).
- Enter latency percentage into the **Accepted latency** entry field. The actual used latency time is shown to the right of the entry field (here 25 ms). The lowest accepted latency time is 10 ms.
- Click **Apply**. Note how the actual latency time changes if the accepted latency percentage exceeds 10 %.
- Click **OK**.



If latency error occurs, tune the tasks. Information about task tuning can be found in the *System 800xA Control AC 800M Planning (3BSE043732*)*.

Latency Alarm Limit

A latency warning is issued if latency is above 70% of accepted latency. A system alarm, actual latency in ms is generated, and added to the system log. A yellow

warning  is written to the Actual column of Latency alarm limit and “Latency high alarm limit exceeded” is written in the Remark field of the task properties dialog box.

Latency is measured on a periodic basic, the time from the start of one execution to the start of next execution is measured. The latency is then calculated as the difference between this value and the interval time. Latency can then be both positive and negative. The maximum latency time is the absolute value of actual latency.



If requested offset is 0 it is possible that actual offset is large, compared to actual latency.

Example

Task A: Interval Time=150 ms, Offset=0 ms, Priority=4 - Low and Execution Time=1 ms.

Task B: Interval Time=150 ms, Offset=0, Priority=3 - Normal and Execution Time=17 ms.

In this case the actual offset of Task A is about 18 ms and actual latency vary from -1 to +1 ms.

The execution of task A is delayed about 18 ms for each interval, which results in an actual offset of 18 ms. This delay is repeated for each period which result in a small actual latency, -1 to +1 ms.

If the interval time of Task A is changed to 50 ms the actual latency of Task A will assume the values -18 ms, 0 ms, +18ms. Actual offset will assume the values 0 ms and 18 ms.

Task Abortion

If a task is aborted, the corresponding application will be stopped. The following criteria apply to a task abortion.

Time-critical Tasks

Time-critical tasks (priority 0) are aborted when the execution time exceeds 300 ms. Time-critical tasks are also aborted if a fatal overrun error occurs. Criteria for fatal overrun errors are set in the Controller Settings dialog box, see also [Overrun](#)

[Supervision](#) on page 179).



In a High Integrity controller running SIL-tasks, error handling is stricter. Compared to a non-SIL application, less severe errors might lead to an application being stopped.

No time critical tasks are allowed in a High Integrity controller.

Non Time-critical Tasks

Non-time-critical tasks (priority 1-5) are aborted when:

- The execution time exceeds 10 seconds.
- The execution time exceeds $(100 * \text{IntervalTime})$.

If overrun supervision is enabled, non-time-critical tasks are also aborted if a fatal overrun error occurs. Criteria for fatal overrun errors are set in the Controller Settings dialog box. See also [Configuring Overrun Supervision](#) on page 179.

This means that if IntervalTime is set to 100 ms or higher ($100 * 100 \text{ ms} = 10 \text{ seconds}$), tasks will be aborted if they have not been executed within 10 seconds.

If IntervalTime has been set to $<100 \text{ ms}$, tasks will be aborted if they are not executed within $(100 * \text{IntervalTime})$.

Load Balancing

The cyclic load is the percentage of controller CPU power used for program execution of application code. If the cyclic load exceeds 70% in the controller, so-called *load balancing* is initiated automatically. The interval time for all tasks, except the time-critical task, is then generally increased, to limit the cyclic load to 70%. Load balancing is not available in High Integrity controllers.

If the cyclic load then falls below 70% again, the interval time will normally be decreased in all tasks, except for the time-critical task. However, the interval time never falls below the original defined interval time.

Whenever the interval time is changed due to load balancing, a *SystemSimpleEvent*, expressed in percent (%) of the actual interval time, is generated, and added to the system log.



Load balancing for the time-critical task is handled as follows (this differs from non-time-critical tasks). The interval time for the time-critical task is increased, whenever its execution time exceeds 50% of its interval time.

For example, if a time-critical task has an interval time of 100 ms, and the execution time becomes 54 ms in an interval, then the new interval time becomes 108 ms. However, the interval time must be reset manually, after it has been increased. The interval time of the time-critical task is never decreased automatically, as for the other tasks.

Change the *Requested Interval Time* to its original value, or another suitable value, in the Task Properties dialog box (in Online mode). Press **Apply** or **OK** to bring the reset into effect.

Whenever the interval time is increased for the time-critical task, due to load balancing, a *SystemSimpleEvent*, expressed as the actual interval time in ms, is generated and added to the system log.

Non-Cyclic Execution in Debug Mode

A task can be set up for non-cyclic execution. Use non-cyclic execution to simplify the debugging of a program.

Debug Mode

Debug mode allows you to debug an application by halting the application running in the controller, and executing the code one execution at the time.

Debug mode is enabled from the Task Properties dialog box (right-click the task in Project Explorer, and select **Properties**).



Tasks marked with SIL cannot be set in Debug mode.

When you have selected *Enable debug mode*, you can halt the cyclic execution of a task by clicking **Halt**. When the task is halted, you can execute the task once by clicking **One Execution**. (This is referred to as “non-cyclic execution”.)

Other tasks will not be affected if one task is set up for Debug mode, they will run in normal cyclic execution mode.

To return to normal cyclic execution of the task, click **Run**.



A task in Debug mode is indicated in Project Explorer with a warning icon (a yellow circle with a black exclamation point).



Functions based on the real-time clock (PID controllers, timers, etc.) cannot be properly debugged in Debug mode.

Timer functions will take into account the actual time elapsed since started, regardless if, for example, the task is halted in Debug mode.

Task Analysis

The Control Builder provides a Task Analysis tool to predict the execution of tasks in controllers before downloading the application to a controller.

The Task Analysis tool provides the following functions before the download of the application:

- Analyzes the task scheduling in the application.
- Presents a graphical representation of how the tasks will execute with the application.
- Detects possible overload situations before the download of the application. The tool detects problems such as task latency, task overrun and overload of task execution.
- Allows remedial actions by providing the option to change the execution time of the tasks and view the updated analysis.



The update of the task execution time using the Task Analysis tool updates the task for analysis only. The actual execution time of the task need to be changed by updating the Task Values in the Task Properties dialog box in Control Builder.

The Task Analysis tool can be used before normal download and before the download using Load Evaluate Go (LEG). For initial download, the execution time of the tasks is assumed to be 1ms for the analysis.

If the task configuration in the Control Builder project is changed before a normal download, the Task Analysis dialog box automatically appears during the normal download. The dialog box does not appear automatically if LEG is used for the download.

To open the Task Analysis dialog box in Control Builder in Offline mode or Online mode, go to **Tools > Task Analysis**.

During download, to enable/disable the Task Analysis function, go to **Tools > Setup > Station > Application Download**, and edit the value in the **EnableTaskAnalysisTool** text box to *true* or *false*. The default is *true*.

Exploring the Interface

The Task Analysis dialog box displays a summary view, a detailed view, and the status of the summary as shown in [Figure 75](#).

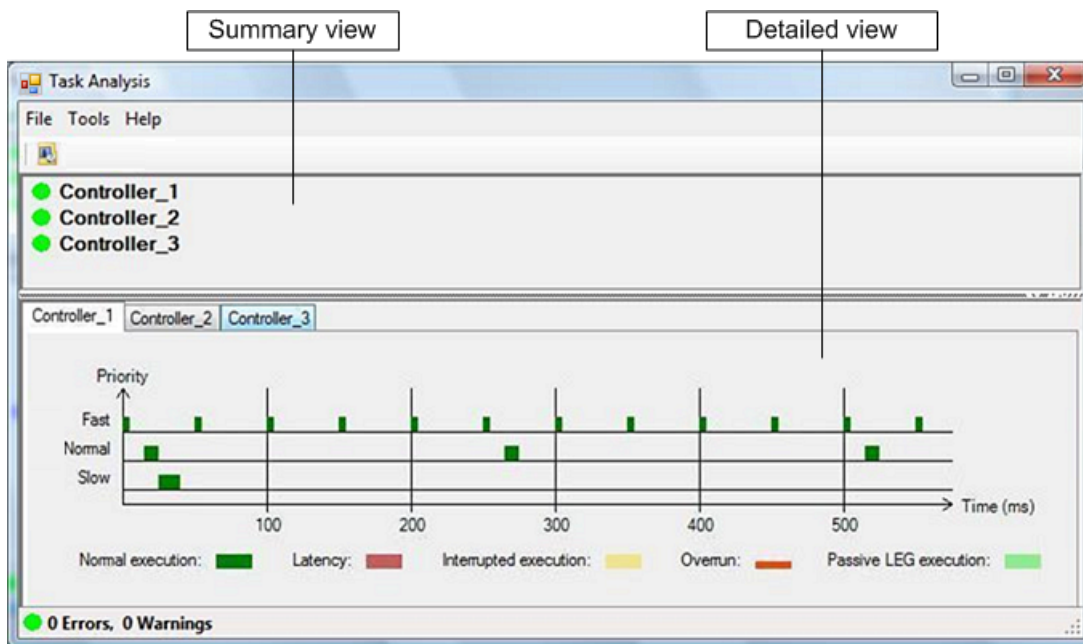


Figure 75. Task analysis tool opened from Tools > Task Analysis

Summary view

This view lists the controllers to which the applications are downloaded. A circular icon (for example, ●) appears beside each controller indicating the various states.

The indications are:

- Red icon: Error
- Yellow icon: Warning
- Green icon: Ok

If the task execution contains errors or warnings, the description of the error or warning is also displayed.

Detailed view

This view displays each controller (listed in Summary view) in a separate tab, as shown in [Figure 75](#).

Click each tab to open the graph showing the task execution of that controller. The tasks are plotted on the graph with the Priority on the Y-axis and the Time (task execution time) on the X-axis. Moving the cursor over each task name opens a tooltip displaying its execution time, interval time, and offset.

Status

This is found at the bottom of the tool interface as shown in [Figure 75](#). It displays the total number of errors and warnings, and the icon (in red, yellow or green). This helps to decide if it is safe to download the new application.

The significance of the indications are described below:

- Red — the new application **cannot** be downloaded as there is a risk of overloading the controller.
- Yellow — the download of the new application **may** cause overloading of controller. The user must, based on analysis, decide if it is feasible to go ahead with the new application download.
- Green — the new application is **safe** to be downloaded to the controller.

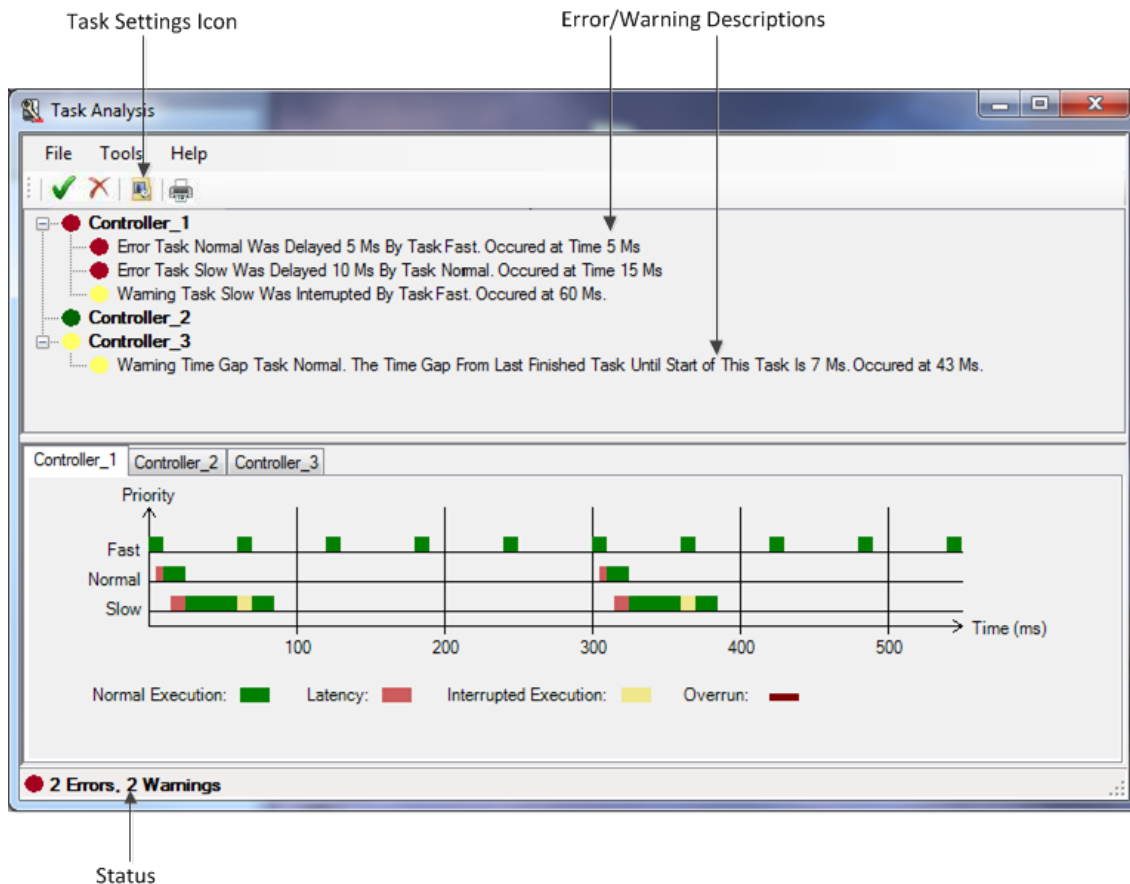


Figure 76. Task Analysis tool with error and warning indications

Task Analysis During Download and LEG Download

If the task configuration in the Control Builder project is changed before the download, the Task Analysis dialog box automatically appears during normal download, with the additional options to accept or cancel the download (✓ and ✗ icons). See Figure 77. These options also appear in the Task Analysis dialog box that is invoked manually during LEG.



If the Task Analysis dialog box shows errors, the ✓ icon is not activated.

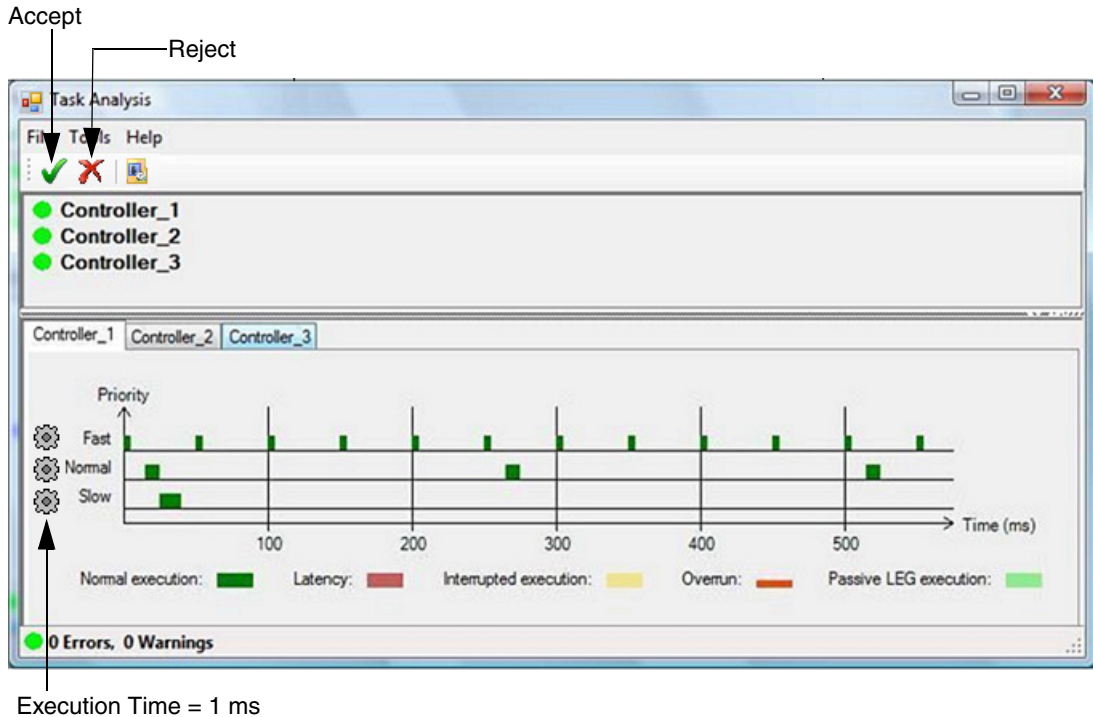


Figure 77. Task Analysis with additional options to accept or cancel the download

The execution time is fetched from the controller. At LEG download, the execution time of the evaluated application is the same as for the old Active application.




If a task is not running in the controller when the execution time is requested, the tool uses 1ms as the execution time, and the ⚙ icon appears beside the task name. See Figure 77. The ⚙ icon also appears if the task execution time is modified for analysis.

Modifying Task Execution Time

The Execution Time of each task can be modified for analysis.

To modify the execution time of a task for analysis:

1. On the detailed view in Task Analysis dialog box, open the tab screen of the controller for which the task need to be changed.
2. Click the  icon, or from the toolbar, select **Tools > Settings**.

The **Task Settings** *ControllerName* dialog box appears as shown in [Figure 78](#).

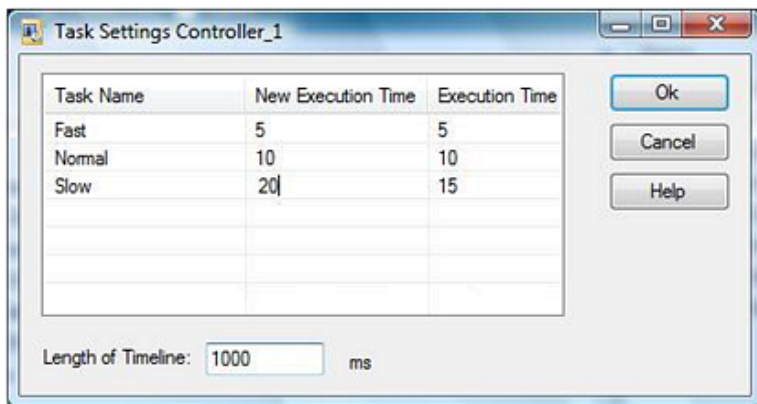



Figure 78. Task settings

3. Modify the execution times under **New Execution Time** column corresponding to the task name, and click **OK**.

The graph is updated as per the new execution time, and the  icon appears beside the task name of the modified task.

Error and Warning Categories

The errors and warnings that are displayed by the Task Analysis tool are generated based on the following categories of analysis:

- Accepted latency
- Task latency
- Task overrun

- Interrupted execution
- Cyclic load overload
- Total load overload
- First scan execution limit
- Too low FDRT
- Internal diagnostics

Table 16 describes these categories and the corresponding reasons for errors and warnings.

In the Task Analysis dialog box, these errors and warnings are displayed with relevant messages that describe the problem.

Table 16. Categories of errors and warnings in Task Analysis tool

Category	Reason for Warning	Reason for Error
Accepted latency	An <i>Accepted Latency</i> value that is set below 10% (default value) of the Interval Time.	-
Task latency	The latency is detected and it is not within the <i>Accepted Latency</i> , but the latency supervision is disabled.	The latency is detected and it is not within the <i>Accepted Latency</i> , and the latency supervision is enabled.
Task Gap	The time gap is too small (less than 20% of the execution time of the task last executed) or ≤ 5 ms.	The time gap is too small (less than 10% of the execution time of the task last executed).
Task overrun	-	The overrun is detected, that is, the task has not finished executing one scan before the next cycle is supposed to start.
Interrupted execution	The task is interrupted by a higher prioritized task, and the task scan is delayed.	-

Table 16. Categories of errors and warnings in Task Analysis tool

Category	Reason for Warning	Reason for Error
Cyclic load overload	The task uses more than 50% of the total cyclic load in the controller. Note: In a HI controller, the recommended maximum cyclic load is 50%.	The task uses more than 70% of the total cyclic load in the controller.
Total load overload	The total load in the controller is above 70%. Note: In a HI controller, the recommended maximum total load is 90%.	The total load in the controller is above 98%. Note: This check is relevant for download using Load Evaluate Go as well as for the download with modified execution times.
First scan execution limit	The load dependent first scan execution time (LFST) is 90% of the maximum FDRT+Accepted Latency, or it is 90% of the maximum Interval Time+Accepted Latency. Tip: The duration of the first scan execution time depends on the cyclic load of the remaining tasks. If the remaining tasks use L% CPU load, the $LFST = (FST * 100) / (100 - L)$.	The load dependent first scan execution time (LFST) is more than the maximum FDRT+Accepted Latency, or it is more than the maximum Interval Time+Accepted Latency.
Too low FDRT	The FDRT value is below 3 seconds.	-
Internal diagnostics	Consecutive task execution is detected for a time that exceeds 40% of the configured FDRT.	Consecutive task execution is detected for a time that exceeds 50% of the configured FDRT.

Security

Security on a type allows the administrator to set permissions for object-specific restrictions like access, download, online changes, etc. This can be done by creating a Security Definition aspect, which allows an administrator to include or exclude user groups on an object level.



If permissions are set on type level, conflicts with general settings might occur for the permission Configure (if the user has the right to configure the object, but not a general permission to configure objects). This will result in an error message when closing the editor after configuring an object. Use default settings for the Configure permission, to avoid conflicts.

The permissions and their related operations are used affects the User Interface in the following way:

Table 17. Permissions and Related Operations

Operation	Permission	Affects
Modify	Configure	Offline editing. Change Settings (system variables) in Control Builder. Changes through Open IF.
Operate	Operator Configure	Online changes. Interacting in CMD (interaction objects).
Deploy	Download	Download. Firmware download. Delete of application in controller.
Acknowledge	Force IO	Online forcing of IO signals. Acknowledge HW Unit errors.
Break Reservation	Break Reservation	Take over reservation of an Entity
SetVariable	Administrate	Change System Variables in the Controller.
Enter	Force SFC	Online changes of SFC code.

For more information, see the *System 800xA Administration and Security (3BSE037410*)*.

Authentication at Download

Control Builder provides authentication at download to controllers from Project Explorer. The user will be prompted for user identification and password before download is allowed.

Enable Authentication at Download

You must have an open control project, in order to enable authentication. To enable authentication for download in Project Explorer:

1. Select **Tools > Setup > Station > Application Download**. The Setup - Application Download dialog box is displayed.

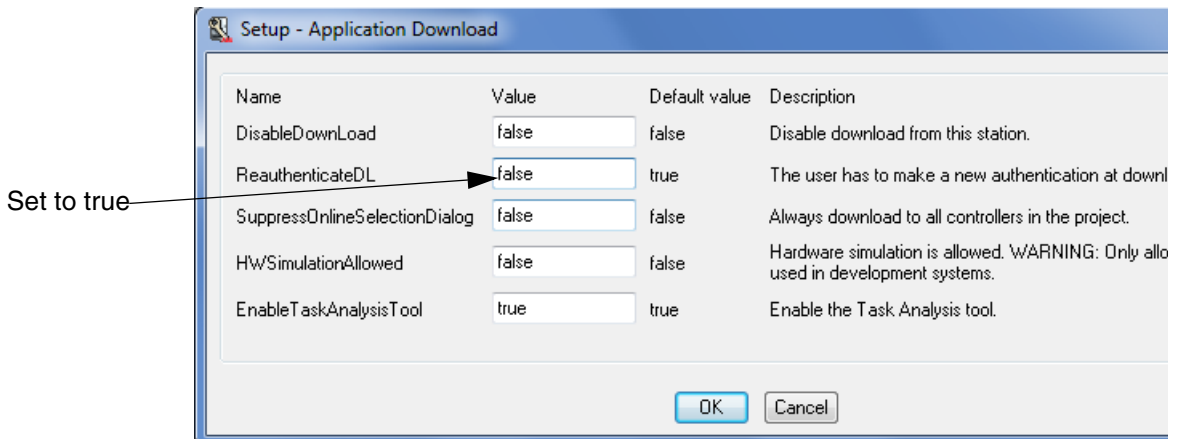


Figure 79. Application Download dialog box for enabling authentication at download.

2. Write *true* in the ReauthenticatedDL field.
3. Click **OK**. Authentication at download is now enabled.

A Reauthenticate dialog box appears before download, see [Figure 80](#).

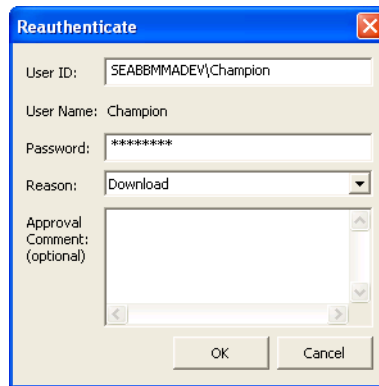


Figure 80. Reauthenticate dialog box, shown before download to a controller.



To disable authentication at download, simply set ReauthenticatedDL to false.

Confirmed Online Write

This subsection describes the Confirmed Online Write function, which is used to configure types and instances in a SIL application, in order for an operator to be able to modify or change values online, and force I/O channels in online mode.

The Confirmed Online Write function is used to set up permissions for writing to SIL application properties online (from the operator workstation). It is necessary to configure the access level for types and instances, in order to make confirmed online write possible.

All changes to protected data will require that the person requesting an online write has the right to make changes. A Confirm Operation dialog box will be shown each time an online write to protected data is attempted.

For more information about Confirmed Online Write see the *System 800xA Administration and Security (3BSE037410*)*.

Search and Navigation

The Search and Navigation function makes it possible for the user to search for symbols (see [Symbol and Definition](#) on page 201) in a project, by using advanced queries, for example, to find out where a certain variable is used in an application.



If a global variable and a data type in the application have the same name the search data base will become faulty. This may result in that a symbol cannot be found.

All symbols matching the search criteria are shown, together with definitions where the symbols are declared. If a symbol is selected, all references where the selected symbol is used in the project are also shown. By double-clicking on a definition, it is possible to navigate to the editor where the symbol is declared. A double-click on a reference shows the editor where the symbol is used.

A report that contains the last search result shown in the Search and Navigation dialog box can also be generated (see [Reports](#) on page 212).




The Search and Navigation function is available in offline, online and test mode. For information on search and navigation in online mode, see [Search and Navigation in Online and Test Mode](#) on page 391.

Search and Navigation Dialog

The Search and Navigation dialog box mainly consists of Search settings, Symbol, Definition and References. All Search settings are remembered and will be applied next time the dialog box is used (until Control Builder is shut down).

The Search and Navigation dialog box can be accessed from Project Explorer, context menus and editors:

- In the Project Explorer, select **Edit > Search**.
- Right-click a Project Explorer object (not Tasks) and select **Search** or **Alt+F12**.
- Select **Edit > Search** or right-click and select **Search** (or **Alt+F12**) in a POU editor, a connection editor, a hardware editor or an access variable editor. These editors also have a search tool bar button  that has the same function.

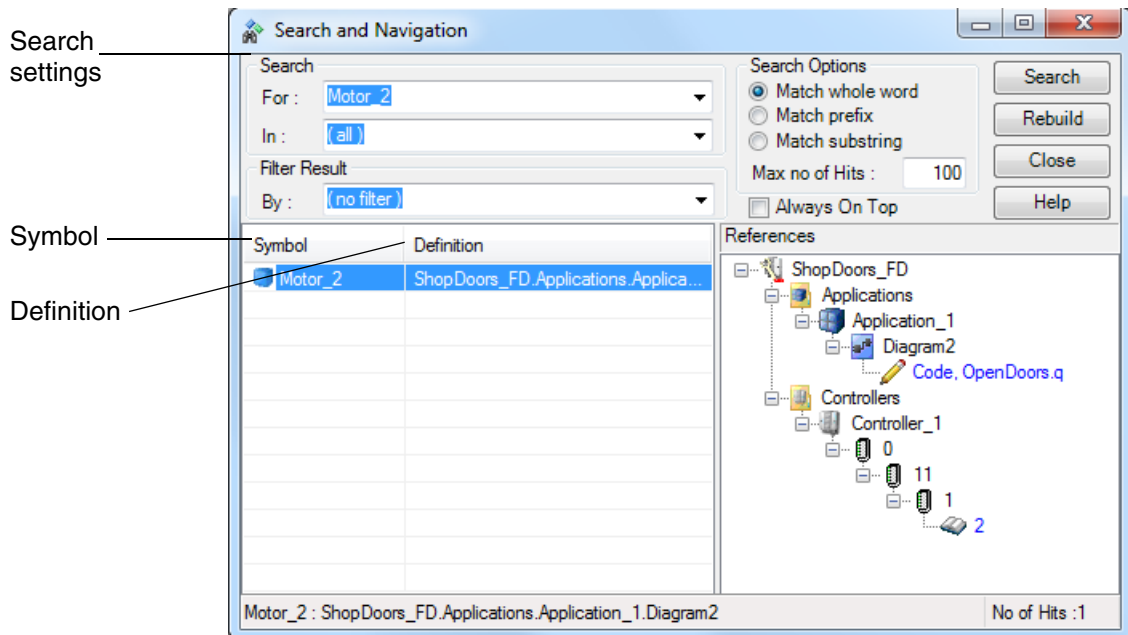


Figure 81. The Search and Navigation dialog box

Search Settings

The Search part of the dialog box consists of the Search For: drop-down list, the Search In: drop-down list, the Search Options radio buttons, the Max no of Hits edit field and the **Search** button. Filter Result belongs to References (see [Filter Result](#) on page 208) and the **Rebuild** button rebuild the Search data base (see [Search Data](#) on page 212).

Search For:

In the Search For text field you enter the symbols to search for (see [Symbol and Definition](#) on page 201). Search Options can be selected for the symbol text entered in the Search For: text field. An empty text or an asterisk (*) character in the Search For: text field search for all symbols. All symbols are case-insensitive, that is, a search for the texts “my”, “My”, “mY” and “MY” gives the same search results.

Search Options

The default setting of Search Options is Match whole word. The Match substring option searches for all symbols containing the entered text as a substring and the Match prefix option searches for all symbols containing the entered text in the beginning of the symbol names.

Max no of Hits:

The entered value in the Max no of Hits: field maximizes the number of symbols that can be found at a search. The default value is 100.

Search In:

The selection in the Search In: drop-down list specifies where, in the project, you want to search for the entered text symbol. An empty text field gives a search through the whole project. Applications, Controllers or Libraries are selected if a search after the Symbol is performed in all applications, all controllers or all libraries respectively.

The text in the [Search and Navigation Dialog](#) on page 199, *Applications.Application_1.Diagram2* performs a search in Diagram2 of Application_1.



In Controllers it is only possible to search for access variables and I/O channels as symbols, since the search symbol has to be defined (declared) under Controllers, in Project Explorer, to match the search criteria.

Select Search “In: Applications” (not Controllers) if you want to know in which I/O unit a certain variable is connected.

Example

In the example below, see [Figure 82](#), a search for the variable “start” is performed to find out which I/O channel it is connected to. “start” is connected to channel 3 in hardware on position 0.11.1. By double-clicking on I/O channel (3), in References pane, you navigate to the I/O unit editor there “start” is connected.

The screenshot shows the Search and Navigation dialog box at the top. The 'Search' section has 'For:' set to 'start' and 'In:' set to 'Applications'. The 'Filter Result' section has 'By:' set to '(no filter)'. The 'Search Options' section has 'Match whole word' selected, 'Match prefix' and 'Match substring' unselected, 'Max no of Hits' set to 100, and 'Always On Top' unselected. Buttons for 'Search', 'Rebuild', 'Close', and 'Help' are visible. Below the dialog box is a table from the Hardware Editor showing I/O channels.

Channel	Name	Type	Variable
QX0.11.1.1	Output 1	BoolIO	Application_1.Diagram2.Motor_1
QX0.11.1.2	Output 2	BoolIO	Application_1.Diagram2.Motor_2
QX0.11.1.3	Output 3	BoolIO	Application_1.Diagram2.start

Figure 82. (Part of Search and Navigation dialog box at top) A search for “start” variable in “Applications” to find out which I/O channel “start” is connected to. (Part of Hardware Editor at bottom).

Search Button

A click on the **Search** button performs the search according to the settings. The search result will be shown.

Always on Top

If Always on Top is checked, the Search and Navigation dialog box is placed in front of all other Windows dialog boxes.

Symbol and Definition

The Symbol objects or the Definitions can be sorted in ascending or descending order, by clicking on the corresponding title. A new click will toggle the sorting order. The selected sorting order is remembered and will be used next time.







Symbol	Definition
 AC800MStatus	ShopDoors_FD.Applications.Application_1.Diagra...
 AppInfoStatus	ShopDoors_FD.Applications.Application_1.Diagra...
 ExtendedStatus	ShopDoors_FD.Applications.Application_1.Diagra...
 LatchedExtendedStatus	ShopDoors_FD.Applications.Application_1.Diagra...
 Status	ShopDoors_FD.Applications.Application_1.Diagra...
 status	ShopDoors_FD.Applications.Application_1.Diagra...

Figure 83. The Symbol and Definition part of the Search and Navigation dialog.

Symbol

A symbol is an object, which can be search for in a project, by using the Search and Navigation dialog box.

Examples of symbols are:

- hardware channels, access variables, project constants, variables, global variables, external variables, communication variables, parameters, extensible parameters, diagrams, programs, function blocks, function block types, control modules, control module types, single control modules, data types, functions, Sequential Function Chart steps, Sequential Function Chart transitions, Sequential Function Chart sequences, applications, controllers and libraries.

Examples of objects that are **not** symbols:

- hardware types, tasks, task connections, comments, descriptions and language statements in the code, labels in Instruction List code, code block names, connected libraries.

A symbol can be selected by clicking on it, clicking on the definition of the symbol or by using the arrow up/down keys on the keyboard.

Definition

The definition of a symbol is where the symbol is declared. The definition of a variable is where in the project the variable is declared, for example in a program.

It is possible to navigate to the definition by double-click on it or by using the context menu. The enter key on the keyboard can also be used. The editor where the symbol is declared is shown with the symbol highlighted.

Definition Context Menu

Right-click a Definition to get the context menu selections.

- **Go To Definition in Editor** navigates to the editor where the symbol is declared.
- **Go To Definition in Project Explorer** navigates to the location of the symbol in Project Explorer.
- **Report...** See [Reports](#) on page 212.

References

The References of a symbol is where in the project the symbol is used.

For example, a variable can be used/accessed by several code lines in several code blocks, and as an actual parameter to a function call or function block call, or as a parameter to a control module/single control module. The variable can also be used (connected to) an I/O channel or an access variable.

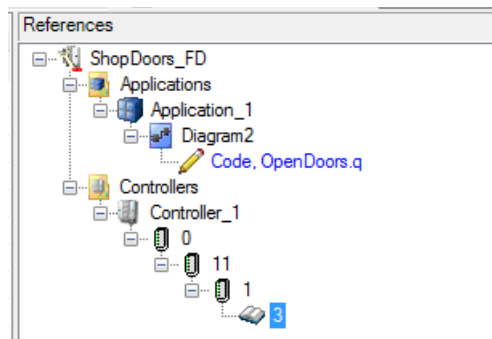


Figure 84. The References part of the Search and Navigation dialog

In the example in [Figure 84](#), the *start* symbol is used at two locations:

- function block OpenDoors, in Code code block of Diagram2.
- in channel 3 of unit 1 at position 0 in Controller_1.

It is possible to navigate to a reference by double-clicking it, or by using the context menu. The enter key of the keyboard can also be used. The present editor is shown with the symbol highlighted.

In the References pane, the project name always appears as the root. Communication Variables are the only symbols that have references in different projects.



If the communication variable that is being searched is present in another project, the References pane displays this variable as part of another project only if the other project also has been downloaded from the same workstation as the current project.

References Context Menu

Right-click on a Reference to get the context menu selections.

- **Go To Reference in Editor** navigates to the editor of the selected reference.
- **Go To Reference in Project Explorer** navigates to the referenced object in the Project Explorer.
- The **Search** menu selection gives the user a possibility to initiate new searches from the references pane. This is useful when a variable/parameter is connected to a parameter of a control module, single control module, function block or diagram type.

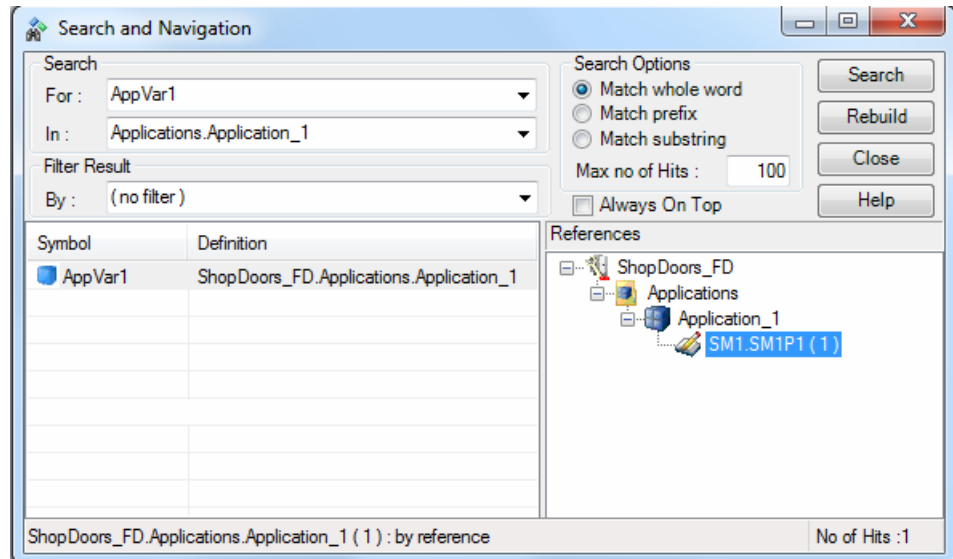


Figure 85. A search for Variable “AppVar1” in Applications.

In the example in [Figure 85](#), *Appvar1* is connected to a parameter *SM1P1* of a Single Control Module named *SM1*.

1. In References, select *SM1.SM1P1(1)*.
2. Right-click and select Search.
The Search For: and Search In: text fields will be automatically updated according to [Figure 86](#). A new search is performed.



The [Execute Search Instantly](#) check box (see [Execute Search Instantly](#) on page 209) has to be checked. If it is not checked, the user must click the **Search** button.

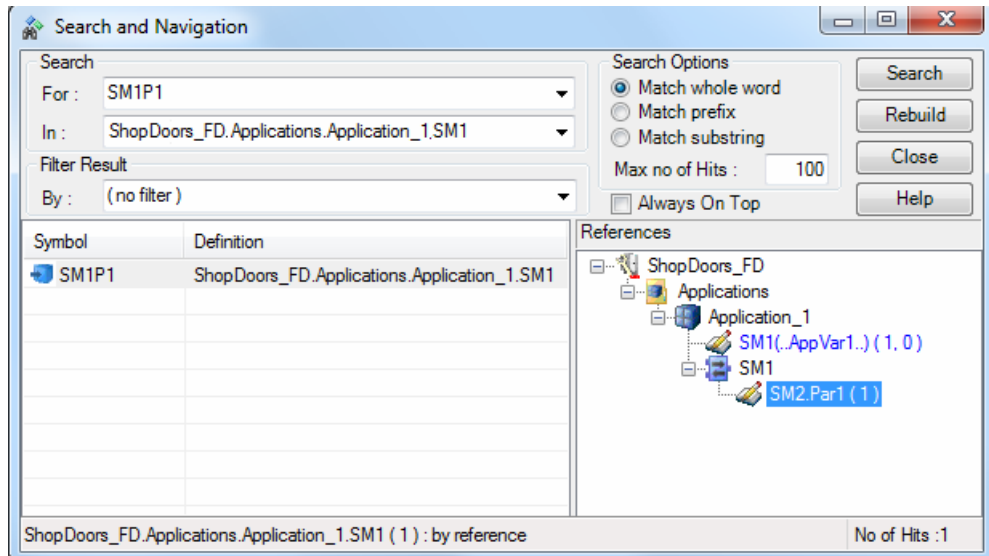


Figure 86. A search for SM1P1 in SM1.

A new search can be done to follow parameter *Par1* in single control module *SM2*.

3. In References, select *SM2.Par1(1)*.
4. Right-click and select **Search**.

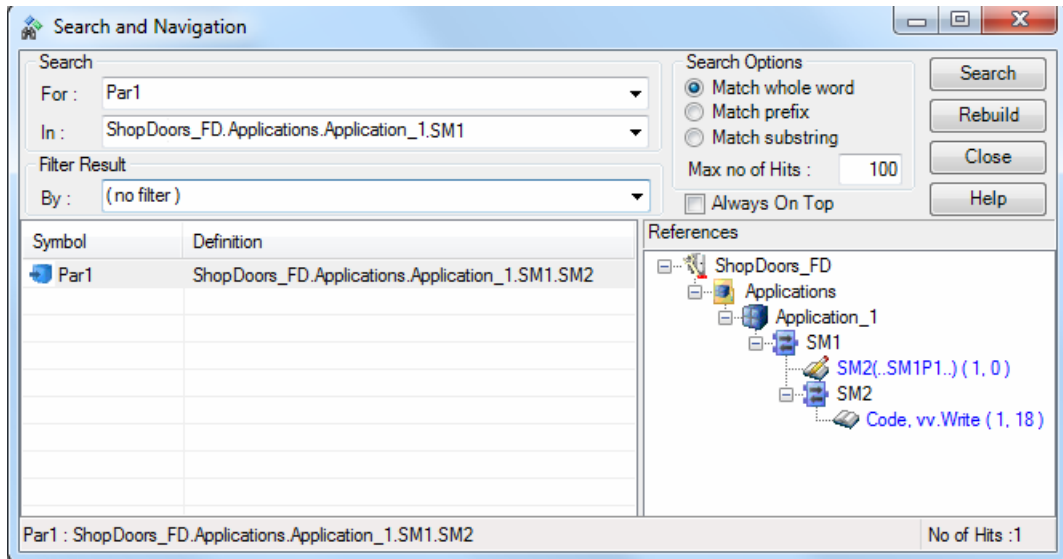







Figure 87. A search for parameter *Par1* in *SM1*.

This example shows an easy way for the user to follow a parameter through a control module hierarchy. The users only have to use the **Search** context menu to follow the parameter downwards the control module hierarchy. It is also possible to follow a parameter upwards a module/function block hierarchy.

Icons in References

The references are marked in blue and preceded by an icon. The icon can be any of the following:

Icon	Description
	The symbol is written.
	The symbol is read.
	The symbol is a function block/function block call.
	The symbol is accessed by reference.
	The symbol is a reference to a graphical connection.

Filter Result

The Filter Result option makes it possible to show references with write access only, or to show references with read access only.

The possible selections are read, write, I/O Channel Out and I/O Channel In. I/O Channel Out shows references to output channels only, and Channel In shows references to input channels only.

Navigation to Editors

It is possible to navigate to the following editors and dialogs:

- The POU editor
- The Connection editor (offline only)
- The Control Module Diagram editor
- The Hardware configuration editor
- The Access Variables editor
- The Project Constant dialog (offline only)
- The Diagram editor

It is also possible to navigate to another project, if it is also displayed in the References pane while searching communication variables.



When navigating to an editor or a dialog box, the window already can be active, but minimized, as well as hidden behind other windows.

It is possible to navigate from a control module parameter or a single control module parameter connection in the References to a Connection editor. However, if the parameter connection is a graphical connection, Control Builder navigates to the Control Module Diagram editor.

Search and Navigation Settings

The Search and Navigation settings dialog box has settings for executing the search and editing of the search fields.

Select **Tools > Setup > Station > Search and Navigation Settings** to view the Search and Navigation settings dialog box.

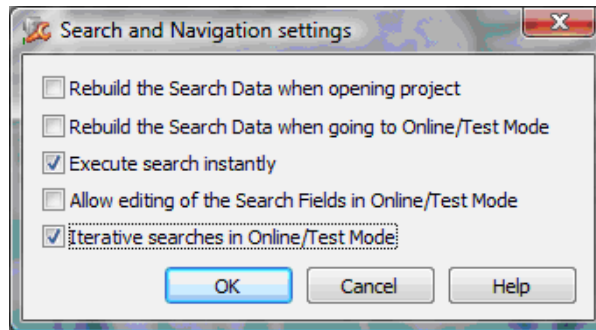


Figure 88. The Search and Navigation settings dialog with default settings

Rebuild the Search Data when Opening Project

When this option is checked, Control Builder will rebuild search data when a new project is loaded in the Control Builder. This check box is, by default, unchecked.

Rebuild the Search Data when Going to Online/Test Mode

When this option is checked, search data is rebuilt when Control Builder is entering online mode or test mode. This setting ensures that the search data is consistent in online and test mode compared to offline mode. This check box is, by default, unchecked.



It is recommended to normally have this check box unchecked

Execute Search Instantly

When this option is checked, the Search and Navigation dialog box will instantly perform a search when the dialog is accessed with the **Search** command, from a menu or tool bar button, that is, the user do not have to press the **Search** button in the dialog. The search is only performed if it is obvious what symbol to search for, that is, both the Search For: and Search In: boxes in the Search and Navigation dialog have to be filled in automatically. This check box is, by default, checked.

Example:

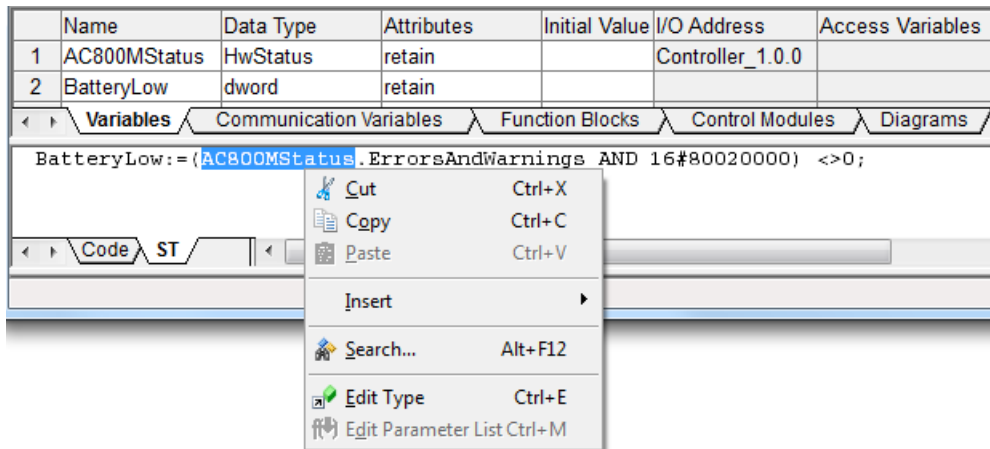


Figure 89. Selection of the AC800MStatus and the Search option

1. In the code block, double click the AC800MStatus variable to select it.
2. Right click and select **Search**, or go to the toolbar and select **Edit > Search** (or **Alt-F12**).

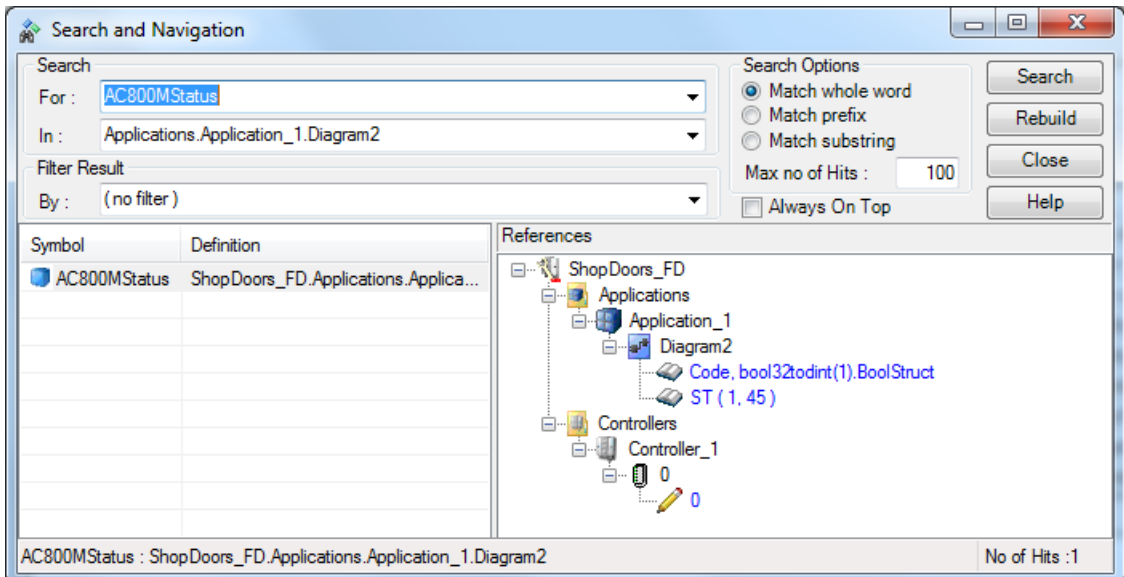


Figure 90. The search result after performing above steps.

Allow editing of the Search Fields in Online/Test Mode

When this option is checked, it enables free editing in the Search field. It is introduced since the strings in the search fields are very sensitive in this mode. A single misplaced character ruins the search and the “search in” field is also case sensitive. This check box is, by default, unchecked.

Iterative searches in Online/Test Mode

When this option is checked, the searches made in Online/Test mode are iterative, and the search hits are presented in one pane. For details, see [Search and Navigation in Online and Test Mode](#) on page 391.

This checkbox is, by default, checked.

Search Data

The Search data base contains search data, that is, information about all symbols, information about the definition of each symbol and information about all references of each symbol.

It is possible to perform a manual rebuild of the Search data base. The Search data base can be rebuilt in the following ways:

- selecting **Rebuild Search Data** from the context menus of application, controller and library.
- selecting **Tools > Rebuild all Search Data**
- clicking the **Rebuild** button in the Search and Navigation dialog

Reports

The search result can be transformed into a report by using *Basic HTML Report.xslt*, that is by default installed together with Control Builder. The report contains the last search result shown in the Search and Navigation dialog. All symbols, definitions and references are included in the report. The symbols in the report are shown in the same order as in the Search and Navigation dialog.

1. Right-click on a Definition and select **Report....**

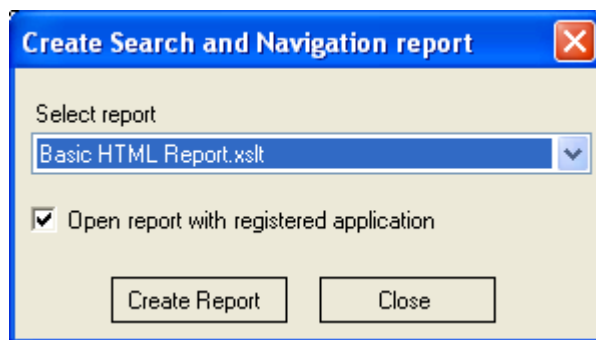


Figure 91. The Create Search and Navigation report dialog.

2. Click **Create Report** button.
If the Open report with registered application is checked, the report will be opened in a registered application. The Basic HTMLReport produces reports in HTML format, that is, the report is opened in the registered Web browser.

3. Specify a directory to save the report in and enter a suitable file name.
4. Click **Save** button to store the report file on disk.

It is possible to export the report to Microsoft Excel by using Export to Microsoft Excel in the Internet Explorer context menu.

Analog Input and Output Signal Handling

Signals start and end in I/O units with I/O channels of the RealIO data type. Between input and output I/O units, signals are handled in I/O function blocks of the RealIO data type, or directly in various function blocks, or in control modules of the ControlConnection data type.

Over and under range measurement

Signal objects of real type are equipped with an option to increase the signal range with a fixed pre-selected factor of $\pm 15\%$ of the specified range. You can select individual Signal Objects connected to variables of data type RealIO on the controller and set the input parameter EnableOverUnderRange to true. The Signal Object enabled with over and under range feature, displays the output parameter OverUnderRangeEnabled as true to inform the surrounding code about the extended range.

Input objects connected to I/O

To enable signal range extensions on input signals, in Project Explorer, click connected **controller** > **Hardware AC 800M** > **Editor** > **Settings**. Set the Clamp Analog in values as false. See [Figure 92](#).

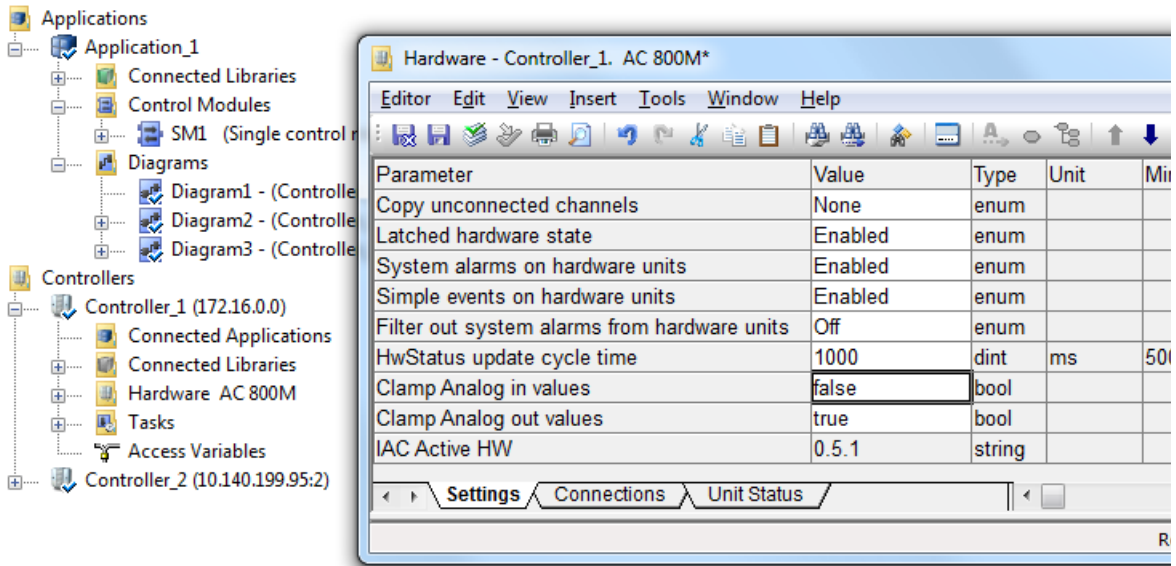


Figure 92. Enabling over and under range for input objects.

Output objects connected to I/O

To enable signal range extensions on output signals, in Project Explorer, click connected **controller** > **Hardware AC 800M** > **Editor** > **Settings**. Set the Clamp

Analog out values as false. See [Figure 93](#).

Parameter	Value	Type	Unit
Copy unconnected channels	None	enum	
Latched hardware state	Enabled	enum	
System alarms on hardware units	Enabled	enum	
Simple events on hardware units	Enabled	enum	
Filter out system alarms from hardware units	Off	enum	
HwStatus update cycle time	1000	dint	ms
Clamp Analog in values	true	bool	
Clamp Analog out values	false	bool	
IAC Active HW	0.5.1	string	

Figure 93. Enabling over and under range for output objects

Backup Media

Backup Media and portable memory cards can be used for several different functions in an AC 800M controller such as transferring data between a Control Builder PC and a controller (that is Controller Firmware or Configuration) and/or for storing data locally in the controller (that is Post Mortem Memory Image or Cold Retain Values). Data stored on the Backup Media does not depend on the controller's battery backup. The Backup Media card can be inserted in a media card reader/writer connected to a Control Builder PC or in the card slot located at the front of an AC 800M controller.

Table 18. Supported Function

Supported Function	AC 800M	AC 800M HI
Dump of Post Mortem Memory Image	Yes	Yes
Upgrading Controller Firmware	Yes	No
Saving Cold Retain Values	Yes ⁽¹⁾	No
Loading Configuration	Yes	No
Storing related Files	Yes	No

(1) Saving Cold Retain values to Backup Media in the controller is not supported for redundant AC 800M controllers.

The functions listed above may all be used together on the same card in a specific controller except for the AC 800M HI controller. However it is not recommended due to the risk of unintentionally activate functions, for example, firmware upgrades, while restarting a controller.



The AC 800M HI controller only supports usage of Backup Media for post mortem memory dump. No files related to other functions may exist on Backup Media used in the AC 800M High Integrity controller.

Card Types

The supported backup media for AC 800M controllers are:

- Compact Flash card (supported in all AC 800M controllers except PM891)
- Secure Digital card (supported only in PM891)



For more information about the AC 800M controller, see the subsection ‘Product Overview’ in the *AC 800M Controller Hardware (3BSE036351*)*.

Compact Flash

Compact Flash (CF) is a portable memory card that can be easily inserted to the card slot located at the front of AC 800M controllers (except PM891).

Specifications for Compact Flash Card. The following are the specifications for the CF card used in AC 800M controllers (PM8xx, except PM891):

- Formatted according to FAT16 or FAT32.
- Minimum read speed – 8MB/second.
- Minimum write speed – 6MB/second.
- Same (or better) ambient temperature operative range compared to the PM8xx that uses the card.



FAT16 is sometimes referred to as just FAT



FAT16 formatting must be used when upgrading controller firmware with a compact flash card. Any size card may be used but cards larger than 2 GB needs to be re-partitioned to less than 2 GB in order to be formatted as FAT16. This will be done automatically if the function Load Firmware to removable media in control builder is used to copy the firmware to the card.

Secure Digital

Secure Digital (SDSC/SDHC/SDXC) is a portable memory card that can be easily inserted to the card slot located at the front of the PM891 controller.



Card type SDSC is sometimes referred to as SD.

Specifications for Secure Digital Card

The specifications for the SDSC/SDHC/SDXC card used in AC 800M controller (PM891):

- Formatted according to FAT32.
- Minimum read speed – 8MB/second.
- Minimum write speed – 6MB/second.
- Same (or better) ambient temperature operative range compared to the PM891 that uses the card.



Cards formatted according to exFAT is not supported.

Adding CF Card or SD Card to Hardware

When a Backup Media card is used in a controller, it is recommended to always configure this in the controller hardware tree, though it is only required by the function Saving Cold Retain Values. The benefit is that system events will be generated when a card is inserted or removed.

Ensure that BasicHwLib is inserted under Hardware and that it is connected to the controller.

From the Project Explorer:

1. Expand the **Controllers** item until you reach the **CF Reader** (or **SD Reader**) item (see [Figure 94](#)).
2. Right-click the **CF Reader** (or **SD Reader**) and select **Insert Unit** from the context menu. A dialog opens.
3. Select **CF Card** (or **SD Card**) in the dialog, and click **Insert**.

4. Click **Close**.

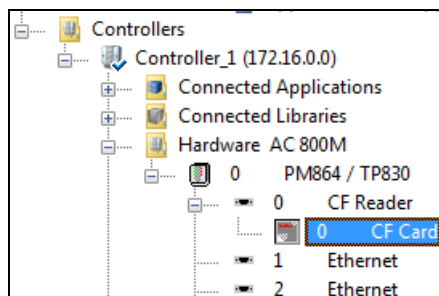


Figure 94. The *Controllers* item expanded and the *CF Card* connected to the *CF Reader* item.

Dump of Post Mortem Memory Image

In most cases of a controller shutdown or crash, the content of the whole RAM memory will be captured and automatically saved on the Backup Media if present. This is valid both for a single/primary PM or a backup PM. This information will, in many cases, greatly increase the probability of finding the root cause of a problem and is added to the error report. See [Error Reports](#) on page 493.



This function is supported in all AC 800M controller types including the AC 800M High Integrity controller.



It is recommended to insert a Backup Media card in case of controller crashes, in order to collect vital information. It is possible to insert the Backup Media card after the crash has occurred. This requires that the Autorestart function (default setting is off) has not been enabled via the IPConfig tool.



The dump file created will always be named DUMP.bin. Hence only one dump file can exist on the Backup Media. Any existing dump file will be overwritten.



It is not recommended to store firmware files on the Backup Media that is used to capture memory dumps. This is in order to avoid initiating a firmware upgrade/installation when restarting a controller.



It is recommended to compress the DUMP.bin file in order to save space before sending it to the support organization.

Ensure that the complete memory dump is saved before the Backup Media card is removed. The status is indicated by the F(ault) LED on the PM8xx, see [Figure 95](#).

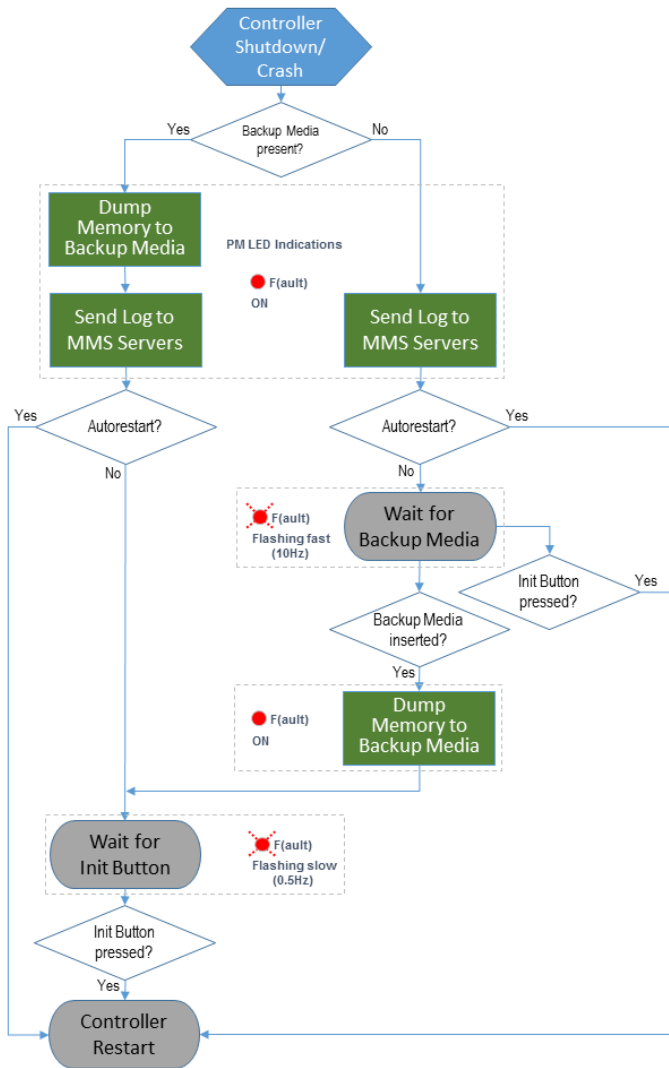


Figure 95. Dump of Memory Image

- F(ault) LED is ON - Dump of memory image in progress. The time needed for the memory dump varies with card type/speed and controller type (memory size).



- A memory dump in a PM891 will take more than 20 minutes. If the controller is manually restarted before the dump is ready the dump file will be incomplete.
- F(ault) LED is Flashing Fast (10 Hz) - Indicating that the controller is waiting for a Backup Media to be inserted after a controller shutdown/crash. If the reason for the shutdown is already known or no Backup Media card is available the controller can be restarted by pressing the INIT button.
 - F(ault) LED is Flashing Slow (0.5Hz) - Dump of memory image is ready. After this the Backup Media card can be safely removed from the controller. The Backup Media card will then contain a file (DUMP.bin) with the total memory content of the controller and in some cases also other log files. The controller can be restarted by pressing the INIT button.

Saving Cold Retain Values on Files

The cold retain values used by the backup media can either be saved cyclically via the settings in the hardware editor, or from the code via the function block (SaveColdRetain).

Either way, these values are only saved on files located on the backup media. Thus, not be confused with the cold retain values saved by Control Builder or OPC Server during a download.



Read more about the SaveColdRetain function block type in Control Builder online help.

An OPC Server will not be able to give any data if the AC 800M starts to execute a different version found at the backup media after the power is resumed. All OPC quality in this case will be BAD, because the OPC Server has no way of finding the correct description files. But, if the backup media has the last created version, then the AC 800M continues to execute the last version, and the OPC server can generate data.

Also note that cold retain values will not be saved on the backup media in case there is an application version mismatch.

Setting Up Cyclic Save of Cold Retain Values

As mentioned earlier, saving cold retain values cyclic are one of two methods for a single CPU configuration. The other method is saving cold retain values based on process events, accomplished by calling the function block (SaveColdRetain) from the code. You should typically decide one of these two methods. However, if you run with a redundant CPU configuration, then you must read [Cold Retain Values for Redundant CPU Configuration](#) on page 223.

This subsection will describe how to save cold retain values cyclic. Provided that you have added the CF Card (or SD Card) to your Hardware tree, do the following:

1. Double-click the **CF Card** (or **SD Card**) and select **Settings** tab in the hardware editor.
2. Set the cyclic interval time for saving cold retain values to file. The default value is (60 min.). See [Figure 96](#).

Parameter	Value	Type	Unit	Min	Max
Save cold retain values	60	dint	min	0	65535

Settings Connections Unit Status

Row 1, Col 2 SCRL

Figure 96. Settings for Save cold retain values (default 60 min.).



To prevent CF card or SD card from saving additional cold retain values, you must set the parameter Value to zero (0). Otherwise it will keep saving new values to file. Setting the value to 0 would normally be the case before shipping the backup media to a host control system.

3. Close the hardware editor.

Cold Retain Values for Redundant CPU Configuration

If you have a redundant CPU configuration; you cannot save cold retain values cyclic or by the function block.

However, you can always save cold retain values via the Tool menu in Control Builder so that your cold retain values will be part of the application, thus be loaded to the backup media.

To save cold retain values for a redundant CPU configuration in Control Builder, first make sure your project is Online:

1. In the Project Explorer menu bar select **Tools > Save “ColdRetain” Values**. A ‘Save “ColdRetain” Values’ dialog will open.
2. Click **Save**. The cold retain values are saved with your application and you are now ready to download to the CF card or SD card. These values will be included when you download the next time to the CF or SD card.



If an AC 800M contains redundant communication interfaces on the CEX-bus, then perform a download to the controller before creating the Compact Flash image. Make sure that the project is not closed while creating the image and before it goes offline, else the image is not completed.

Downloading the Application to Removable Media

Before you can download your application to the backup media, you must connect an external Compact Flash Writer or Secure Digital Writer to your Control Builder PC. The writer is normally connected to the PC's USB port.



It is not possible to download to removable media if the Difference Report is enabled for the project. To check the Difference Report setting, right click the project name, and select **Settings > Difference Report**.

From the Project Explorer, make sure your project is in offline mode and that the Difference Report is not enabled. Then, do the following:

1. Insert a Compact Flash card or a Secure Digital card in the Writer slot.
2. Right-click controller and select **Download to Removable Media** from the context-menu. A **Backup Media** dialog window will open.
3. Select Writer and click **OK**. The Control Builder will write the application to the backup media.



In case the Control Builder source code files is to be placed on the CF/SD card, it is recommended to zip these files into one single file before placing it on the card.



For a redundant CPU configuration, you need to write the same application twice (two CF/SD cards, one in each CPU). Copy (in Windows Explorer) the downloaded application (two folders) from the CF/SD card and paste them temporarily on your local disk. Insert the next memory card into the Writer and drag your two folders from the hard disk and drop them on the new CF memory card.

Configuration Load

A controller will load the configuration (Application and Controller Configuration) and cold retain values from backup media during the following circumstances:

- During controller power-up if it is detected that the content of the memory is corrupted or the battery status is bad.
- During a controller reset.

See flow chart [Controller Restart Modes and Backup Media Usage](#) on page 231 for details.



Loading the configuration from Backup Media is not supported for AC 800M High Integrity Controllers.



For redundant AC 800M controllers it is recommended to insert a Backup Media card in both processor modules. Both cards must contain the same configuration version.



Loading the configuration from Backup Media is not supported when using distributed applications.



This function is not intended to be used in a DCS system. A typical installation where this function is used is a stand-alone controller, often a remote installation, without battery backup.



An AC 800M configured as time master (CNCP order number 1) does not transmit any clock synchronization messages if it starts from a backup media image, and the time quality in the AC 800M is bad due to a discharged battery. The time in the AC 800M has to be manually set using the function block SetDT in order to have the clock synchronization in place.

Application Version Check

If the application version in the controller is not identical with the version in the backup media or vice versa; a warning message will alert and no more cold retain values can be saved.

Upgrading Controller Firmware using Backup Media

A controller may be upgraded with new firmware from the Backup Media during a manually initiated Controller Reset (a long press on the INIT button). If a card is present, then the controller checks for a valid firmware in it. If a valid firmware is found, it will be used for upgrading the current firmware. See flow chart [Controller Restart Modes and Backup Media Usage](#) on page 231 for details.



Firmware upgrade is not supported for AC 800M High Integrity controllers.



For PM85x-PM86x controllers only FAT16 formatted CompactFlash cards are supported. Due to this the maximum partition size is 2 GB. Larger CompactFlash cards will need to be re-partitioned. This restriction does not apply for the PM891. This will be done automatically if the function Load Firmware to Removable Media in Control Builder is used to copy the firmware to the card.



Its only the PM firmware that may be loaded from a Backup Media card; hence firmware for CI modules (e.g. CI854) must be downloaded from the Control Builder.



A firmware upgrade is never done automatically during a normal start-up, a long press on the Init-button is required as stated above.



The Control Builder function Load Firmware to Removable Media will prepare the card if needed and copy necessary files to the card.



The firmware upgrade function in PM85x-PM86x controllers uses a low level function to locate a special “boot” file on the CompactFlash card which does not depend on the normal file system. Hence it may find this file even if it has been deleted unless a thorough reformatting has been done. See chapter [Remove Files Completely from a CompactFlash Card](#) on page 233 for further details.

Upgrading a controller’s firmware using a removable backup media, involves the following steps:

1. Loading a copy of the firmware (that is, a firmware image) onto the backup media using Control Builder (refer [Loading the Firmware Image to Removable Media](#) on page 226).
2. Upgrading the controller firmware using the image on the backup media (refer [Upgrading Controller Firmware from a CF/SD card](#) on page 230).

Loading the Firmware Image to Removable Media

Follow the steps given below for loading a firmware image from the Control Builder to a removable backup media:

1. Mount the backup media card (SD or CF card) on the card reader-writer of the Control Builder PC. Make sure that no other program uses or accesses the card.
2. Right-click on the controller object of the same type as the controller to be upgraded.

3. From the context-menu, select **Load Firmware to Removable Media**. The Load Firmware to Removable Media window appears.
4. The Load Firmware to Removable Media window displays details of the card being used and the action that will be taken. The displayed details differ depending on whether the media card is SD or CF. Click **Yes** to proceed or **No** to cancel the operation.

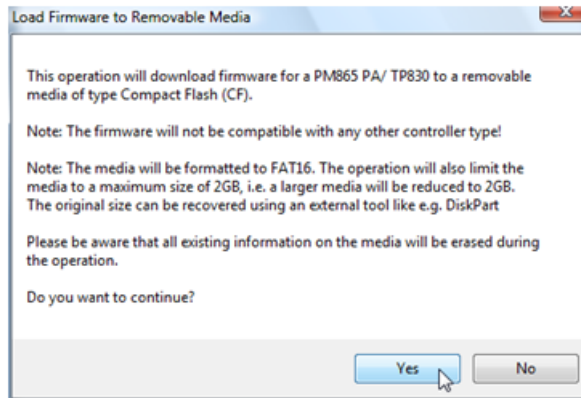


Figure 97. Card details for PM865 PA/TP830

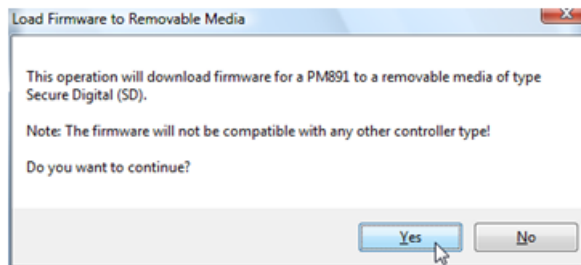


Figure 98. Card details for PM891

5. On clicking **Yes**, a list of identified removable media appears. If the card is not present in the list of removable media, then try the following:
 - Unmount and then remount the media again.
 - Make sure that the card is formatted in a file system. If not, use the Windows format tool or Diskpart to format it in FAT or FAT32 file system.
6. Click the required removable media to select it, and then click **Yes** to proceed. Either of the following cases will happen:
 - If the media used is CF, the Diskpart tool then formats the CF card to FAT 16 with a maximum size of 2 GB (even if the size of the card is greater). The progress of the Diskpart tool will be displayed in a command prompt window. Upon completion the window closes automatically and the firmware image is copied to the card
 - If the media is an SD card, then no formatting is required at this point. The firmware image is copied to the card.

If the above operations are a success, then:

- There will be four files on the card (see [Figure 99](#) for CF card and [Figure 100](#) for SD card). If the media card is CF, then it has been formatted as FAT.
- The file *content.txt* has been rewritten and the first row describes the selected controller. Other rows remain either untouched or partly rewritten.

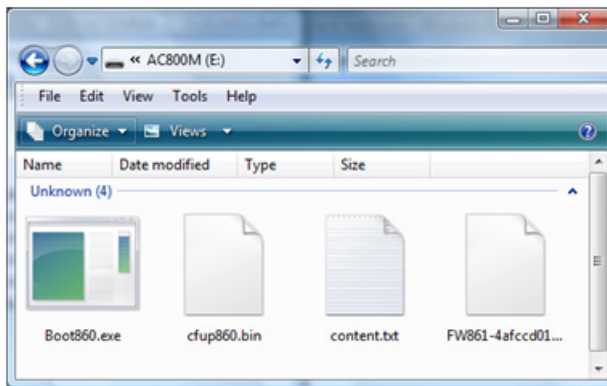


Figure 99. Firmware image files on CF card

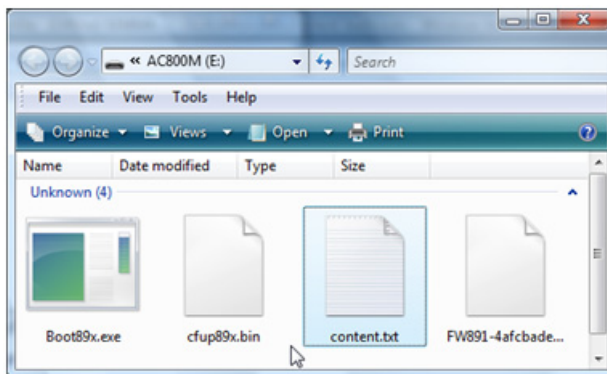


Figure 100. Firmware image files on SD card

If the operation is a failure, a message is displayed conveying the same. Further information about the failure can be found in the Control Builder session log.



While Diskpart is formatting the CF card, it is possible that Windows may discover the card as an unformatted disk. In such a case, the following dialogue will be displayed. Here, select **Cancel**. If not, there will be two program instances trying to format the card at the same time.

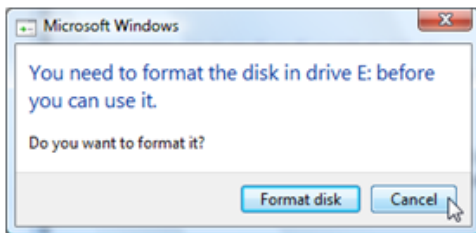


Figure 101. Windows dialog for formatting unformatted disks

Upgrading Controller Firmware from a CF/SD card

1. Insert the CF card or SD card in the card reader slot of the CPU and Power-On.
2. Perform a Controller Reset, by pressing and holding the INIT button till the green Run LED starts flashing.
3. Release the INIT button to start the loading firmware. The process to load the Firmware starts, and the F(ault) and the B(attery) LEDs indicate the progress.

At the end of the operation, the hardware reset starts the newly programmed system.

Controller Restart Modes and Backup Media Usage

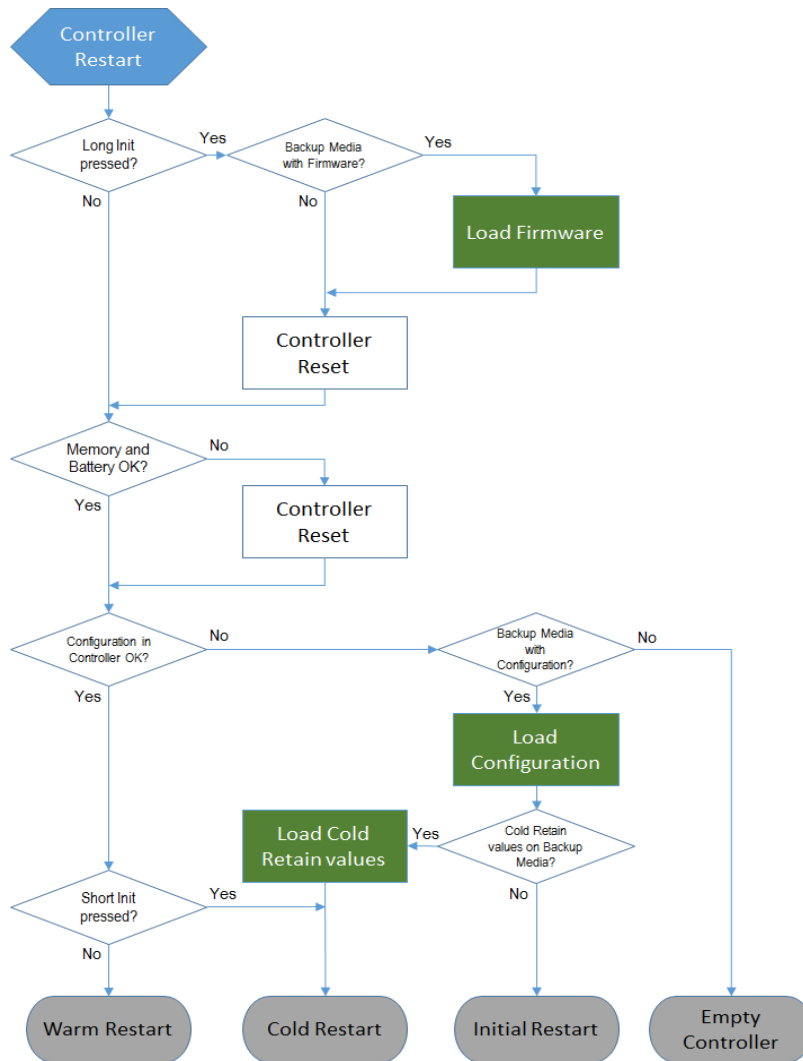


Figure 102. Controller Restart Modes and Backup Media Usage

Storing Related Files

It is possible to store related files on the Backup Media.



Storing related files on Backup Media is not supported for AC 800M High Integrity Controllers.

Restoring Formatted CF Cards to Original Size

In some cases the CF card used for copying the firmware from Control Builder, may have a size of more than 2GB. During copying of the firmware, Diskpart will format this card to FAT 16, and the size will be limited to 2GB. In such a case, to restore the card to its full original size, follow the steps given below:

1. Start the Diskpart tool by selecting **Run** and then type **diskpart.exe**. The Diskpart tool opens in the command prompt. You can then proceed with the commands written in bold in the following steps.
2. List all volumes to identify the actual media by using the **list volume** command.
3. Select the volume by its number or drive letter *n* by using the command **select volume n**.
4. Clean the selected volume using the command **clean**.
5. Create a partition using **create partition primary** command.
6. List all partitions to identify the partition by using **list partition** command.
7. Select the partition by its number *n* by using the command **select partition n**.
8. Activate the partition using the **active** command.
9. Proceed to format the card with default settings using the **format** command.
10. When the format is complete, **exit**.



You may try the above steps in also cases where the card is shown as unformatted and/or unreadable.

Remove Files Completely from a CompactFlash Card

To completely delete all the files from a CompactFlash card, that is been used to upgrade firmware, in order not to accidentally initiate a new upgrade of a PM85x-PM86x controller use the following steps.

- Insert the CompactFlash card in a card reader/writer connected to a PC.
- Right click the drive in the Windows Explorer window and Select “Format...” and ensure that “Quick Format” is not selected under Format options.

Compiler Switches

Compiler Switches are used to control the behavior of the compiler by setting additionally language restrictions.

Global restrictions are valid for all code. Restrictions can be set to generate errors or warnings at compilation. For SIL applications it is also possible to set additionally compiler restrictions. At compilation, errors and warnings are generated according to these settings and global restrictions. These restrictions can be used to stop the use of complex constructions in code, which might cause instabilities or errors.

Global restrictions and SIL restrictions are combined as follows:

- A global error and a SIL warning always generate an error
- A global warning and a SIL error generate a warning for non-SIL applications and an error for SIL applications.

It is possible to exclude a library from checking with user-defined compiler switches. Only warnings can be excluded for a library, not errors.



If a library is excluded from a certain restriction, this restriction will not be checked for any type belonging to that library.

If restrictions are changed, a re-compilation is required before the next download.

Restrictions are checked both at compilation and when checking the code.

Settings

Right-click the control project (root object) and select **Settings > Compiler Switches** to open the Compiler Switches dialog.

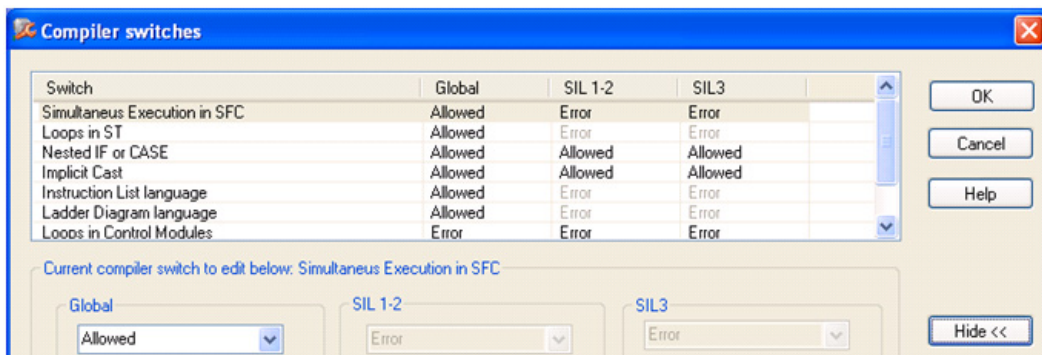


Figure 103. Compiler Switches dialog

The possible settings of compiler switches are described in [Table 19](#).

Table 19. Compiler Switches

Switch	Description	Global (Non-SIL)	SIL 1-2	SIL 3
Simultaneous Execution in SFC ⁽¹⁾	Simultaneous sequences in SFC	A	E & M	E & M
Loops In ST	Loops in Structured Text (FOR, WHILE, REPEAT and EXIT)	A	E & M	E & M
Nested IF or CASE	Nested IF and CASE statements in Structured Text	A	A	A
Implicit Cast ⁽²⁾	Automatic conversion of data types (e.g. integer to real)	A	E	E
Instruction List language	Instruction List	A	E & M	E & M
Ladder Diagram language	Ladder Diagram	A	E & M	E & M
Loops in Control Modules ⁽¹⁾	Code sorting loops	E	E	E
SFC Language	Sequential Function Chart Language	A	A	E & M

Table 19. Compiler Switches (Continued)

Switch	Description	Global (Non-SIL)	SIL 1-2	SIL 3
Force I/O from code	The compiler switch for forcing I/O signals from 1131 code. It restricts changes of the Forced component in variables of one of the data types BoolIO, RealIO, DintIO and DwordIO and results in either warning or error when the switch is activated. Example: MyBoolIOVar.Forced := true; Not allowed since this assignment directly affects the "Forced" component.	A	E & M	E & M
Multiple calls to the same Function Block	This switch defines if the compiler should check if a POU Type contains more than one call to a specific Function Block instance	W	W	E
None or multiple calls to ExecuteControlModules	The ExecuteControlModules function is called once in every scan from a Function Block Type that contains ControlModule instances. This switch decides if the compiler checks that the call is made correctly.	W	W	E
Non-SIL communication variables in SIL1-3 applications	Restricts the possibility to use Non-SIL classified communication variables in a SIL1-3 classified application.	n/a	E	E

Table 19. Compiler Switches (Continued)

Switch	Description	Global (Non-SIL)	SIL 1-2	SIL 3
SIL 1-2 communication variables in SIL3 applications	Restricts the possibility to use SIL1-2 classified communication variables in a SIL3 classified application.	n/a	n/a	E
No forward flow between blocks in FD	Verification of forward flow in Function Diagram	W	E	E
Max four lower SIL communication variables	Check that max four communication variables with lower expected SIL is used	A	AEW	AEW

- (1) This switch does not affect the "sequence selection" functionality of SFC.
(2) In SIL applications it is recommended to set this switch to Error.

Notes to [Table 19](#)

“**A**”: - **Allowed**, Gives no error or warning

“**W**”: - Gives a compiler **Warning** if the rule is violated, acknowledge required before download is allowed.

“**E**”: - Gives an compiler **Error** if the rule is violated, download is blocked.

The default settings are marked with boldface letters in the table.

“**E & M**” - **Error and mandatory**, same as "Error" but can not be changed by the user.



See Control Builder online help for more specific information how to configure compiler switches.

Reports

Difference Report

If the Difference Report function is enabled, the Difference Report Before Download dialog displays (in the same dialog):

- Differences
- Source Code Report
- Compilation Messages

Based on the information presented in the reports, you can either accept or reject the changes, to continue with the download or cancel the download.

To enable/disable the function, right-click the control project folder (root object) and select **Settings > Difference Report**. This setting is only available for non-High Integrity controllers, whereas the Difference Report appears automatically while downloading to High Integrity controllers.



The Difference Report and start value analysis functions is always enabled when downloading SIL applications to High Integrity controllers.

Difference report shows the difference between data downloaded to the controller and the data present in Control Builder, see [Figure 104](#). The tree view to the left shows the parts of the application that have changed. By clicking an item in the tree, you can display the present controller code to the left, and the new code to the right. Differences are also indicated by colors (the color coding is explained on the status bar at the bottom of the report window).

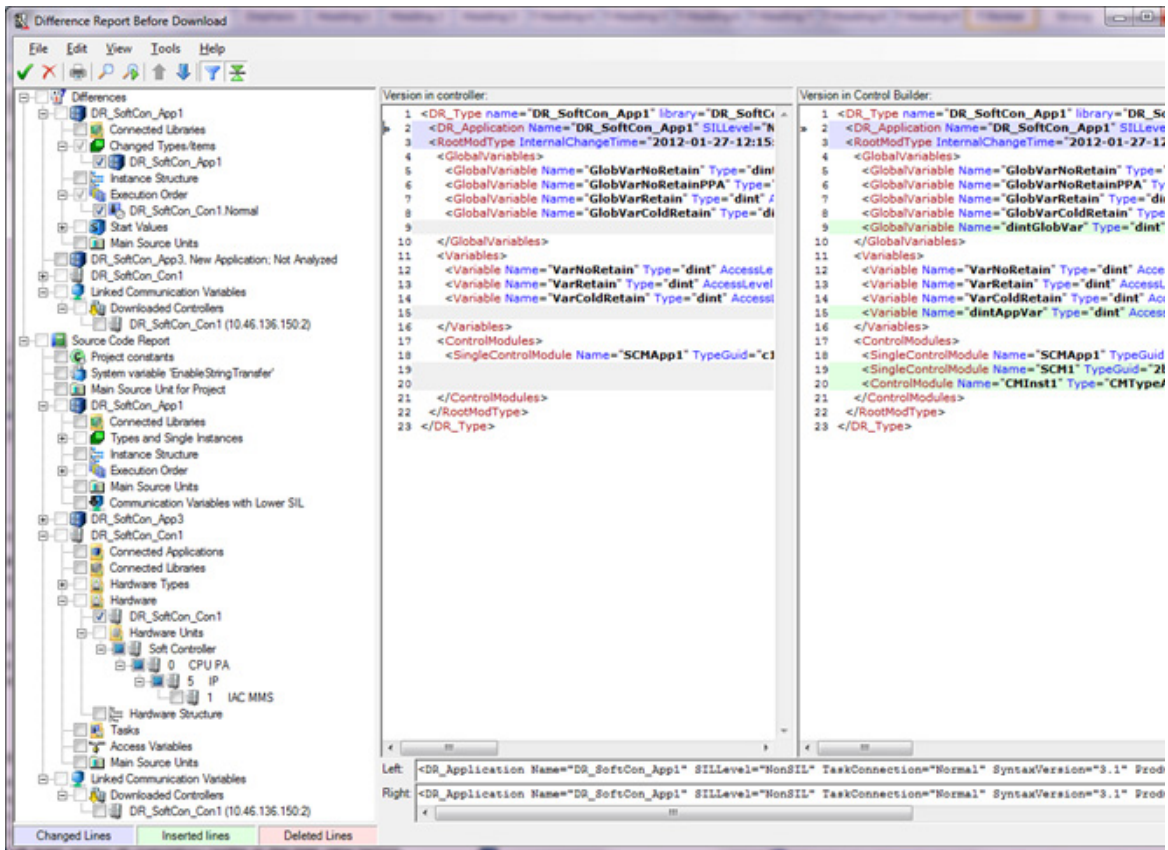


Figure 104. Difference Report Before Download

Review of Mandatory Items Before Downloading to HI Controllers

In the Difference Report, there are mandatory items that should be reviewed and acknowledged before downloading to a HI controller. If there are unchecked mandatory items when you accept the report, a warning dialog appears. This dialog also displays the list of mandatory items that are not reviewed (unchecked items during the review).

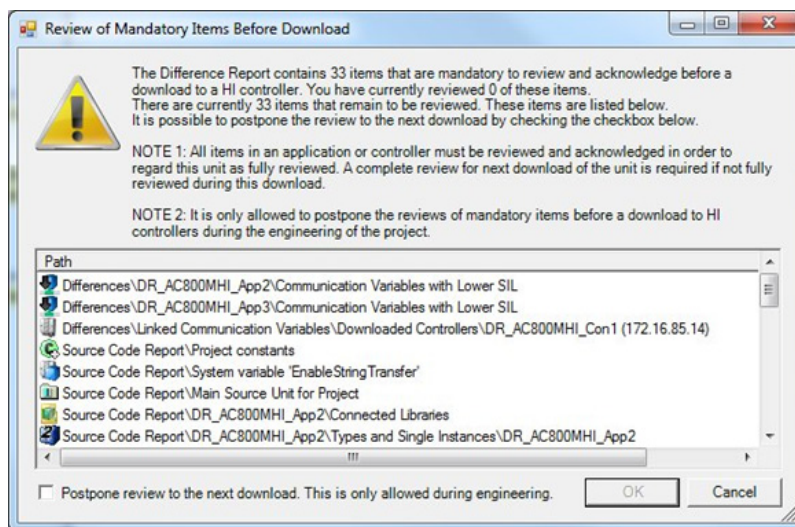


Figure 105. Example of Review of Mandatory Items Before Download dialog

In this dialog, it is possible to postpone this review to the next download if the download occurs during engineering or testing. Click the check box **Postpone review to the next download**, if required.

But, it is not allowed to postpone this review before downloading to a High Integrity controller in a production site. Refer to *System 800xA Safety AC 800M High Integrity Safety Manual (3BNP004865*)*.

It is also possible to enable/disable the postponing of review of mandatory items for SIL applications. To enable/disable the function:

1. Right-click the control project folder (root object), and select **Settings > Difference Report**.
2. In the Difference Report Settings dialog, use the **Allow postponing of mandatory items** check box, to enable/disable the postponing of review of mandatory items (see [Figure 106](#)).

Configuration

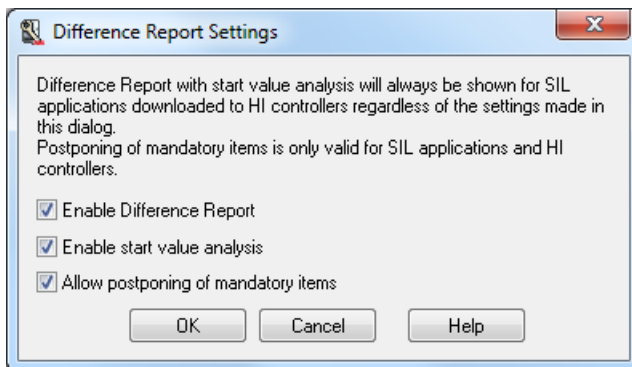


Figure 106. Difference Report Settings dialog

The Difference Report, shown during the download sequence is configurable on the project level.

Difference Report has three configuration options:

1. Enable Difference Report:

If Difference Report is disabled, only HI-controllers and its applications will be analyzed by Difference Report. The disabled controllers/applications will be marked as disabled in the Difference Report User Interface. If Difference Report is disabled the Source Code Report for Controller will be disabled as well when a new download have been done.

2. Enable start values analysis:

If disabled, the start values analysis will be disabled during Difference Report but the rest of the report will be presented to the user. The start value analysis is always enabled for applications executing in HI-controllers. The disabled start values are marked in the User Interface.



Only available if Difference Report is enabled.

3. Allow postponing of mandatory settings:

If set, the postpone checkbox in the “Review of Mandatory Items” dialog will be enabled, making it possible to postpone the review of mandatory items (see Review of Mandatory Items Before Downloading to HI Controllers on page 238 for more information).



Only available in Control Builder and if Difference Report is enabled.

The Difference Report is mandatory for HI-controllers and all applications executing in the HI-controllers.

The Difference Report is also shown during transfer to Test Mode. If Difference Report is disabled, no Difference Report is shown, even for HI-controllers. Both Difference Report and Start Value analysis is by default enabled when a new HI-project is created and disabled when a Non-SIL project is created. Postponing of mandatory items is allowed by default.

The configuration option does not affect Source Code Report when generated in offline mode from the Tools menu. This feature is always available.

The Difference Report presents the differences found, under the *Differences* item in the tree view.

Table 20. Categories under 'Differences' item

Category	Displays Differences for...	Mandatory to Check in SIL or Not.....
<i>ApplicationName</i>	<ul style="list-style-type: none"> • Project Constants • Connected Libraries • Changed Types/Items • Instance Structure • Execution Order • Start Values • Main Source Units • Communication Variables with Lower SIL <ul style="list-style-type: none"> – Communication variables in the application, with the <i>Expected SIL</i> property set to a value lower than the SIL of the application 	All items are mandatory, except Start Values
<i>ControllerName</i>	<ul style="list-style-type: none"> • Connected Applications • Connected Libraries • Hardware Types • Hardware • Tasks • Access Variables • Main Source Units • Foundation Fieldbus <ul style="list-style-type: none"> – If an external configuration tool like Fieldbus Builder FF is used, the changed settings are displayed 	All items are mandatory

Table 20. Categories under 'Differences' item (Continued)

Category	Displays Differences for...	Mandatory to Check in SIL or Not.....
Linked Communication Variables	<ul style="list-style-type: none"> Downloaded Controllers – Communication variables in downloaded controllers Resolved Controllers – Controllers where communication variables will be resolved as a result of the download 	All items are mandatory
System variable <i>VariableName</i>	Changed System Variable	Mandatory



To reduce the compilation time during download of a project to a controller, it is possible to exclude the start values from the difference report. The start value analysis is enabled/disabled via **Project > Settings > Difference Report**.

The start value analysis cannot be disabled for a High Integrity controller.

History of Difference Report

The accepted Difference Reports could be accessed again after a download is conducted. Select **View Accepted Difference Reports** from **Tools** menu to view the list of reports with date and time of download.

Printing Difference Report as a PDF File

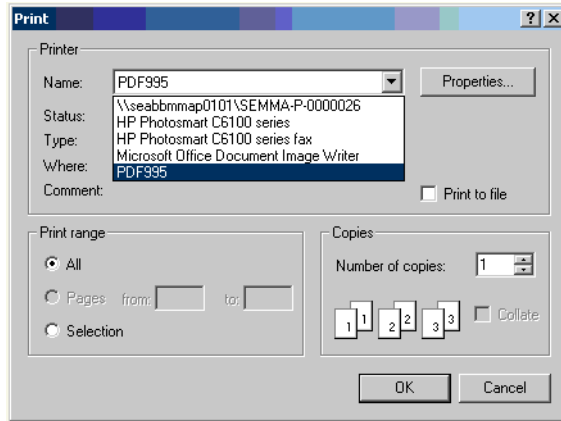
It is possible to print to a PDF-file instead of a printer if a PDF printer driver (Adobe PDF or PDF995 or others) and the corresponding converter is installed.

To print the Difference Report as PDF File:

1. Select **File > Print** in the Difference Report.



2. Select **PDF995/Adobe PDF** in the Print Dialog. Click **OK**.



3. A Save As dialog displays. Enter the file name/folder and click **Save**.
4. The PDF Viewer application is launched to display the difference report in PDF format.

Difference Report Viewer

Difference Report Viewer is a separate executable that can be installed on a PC without the Control Builder. It is available on the installation media, ...*Engineering\Control Builder MTools\Stand Alone Difference Report Viewer*.

The Difference Report Viewer enables the possibility to read historically accepted difference reports in the same format as they were opened from Control Builder M Professional, see [History of Difference Report](#) on page 243.

Launching the executable brings up a dialog with a file browser where the user can browse for the file to view.

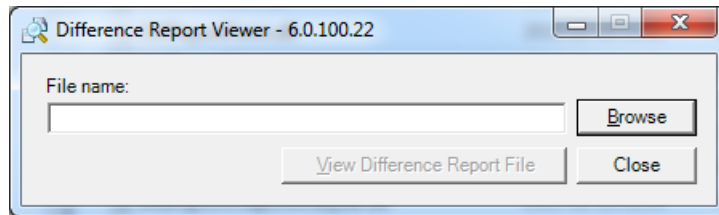


Figure 107. Difference Report Viewer

The **Browse** opens standard file browser that fills in a file name in the text field. The **View Difference Report File** is enabled when the text field contains a valid file name, and brings up the Difference Report user interface.

Source Code Report

The source code report shows the complete source code for the current project in the Control Builder, and enables a review of the source code that is independent of editors and user interfaces of the Control Builder.



The source code report is mainly used for High Integrity applications, where it is important to verify application and controller configuration. The source code report is particularly useful the first time a project is downloaded, when the difference report contains no information.

You perform the review by comparing the code presented in the report with the code in the editors of the Control Builder, checking that the source codes correspond with each other. If you find discrepancies, for example in the controller configuration, you can try to compile and download again.

The main difference compared with the difference report is that the source code report shows all source code from the different parts.

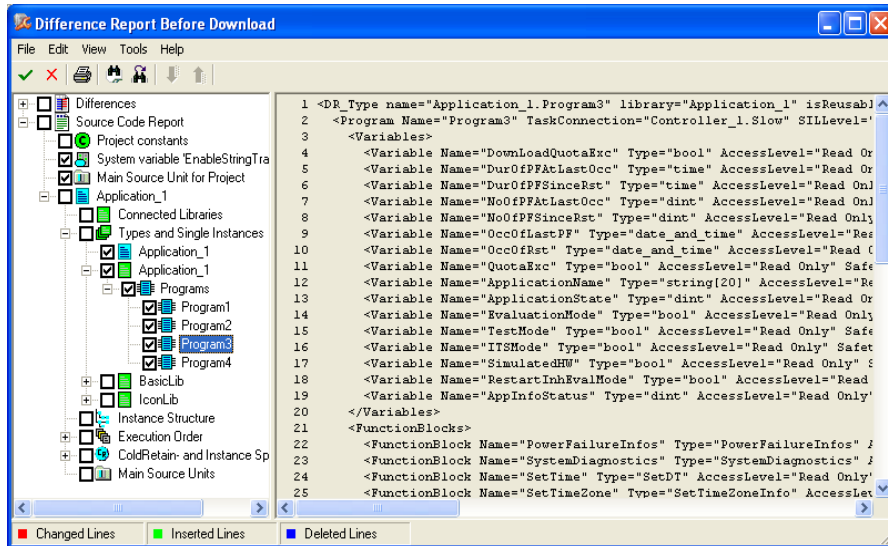



Figure 108. Source code report before download

The left part of the dialog displays a tree containing the different parts of the report (see table below). To view the source code for a specific item, navigate the tree until you find the item, and then double-click the item (or right-click the item and select **Show Source Code**).

Table 21. Categories under 'Source Code Report' item

Category	Displays Source Code for...	Mandatory to Check in SIL or Not...
Project Constants	Project Constants	Mandatory
System Variable <i>VariableName</i>	System Variable	Mandatory
Main Source Units for Project	Main source units for project	Mandatory
<i>ApplicationName</i>	<ul style="list-style-type: none"> • Connected Libraries • Types and Single Instances • Instance Structure • Execution Order • ColdRetain and Instance Specific Values • Main Source Units • Communication Variables with Lower SIL 	<p>All items are mandatory, except ColdRetain and Instance Specific Values.</p> <p>The item – Communication Variables with Lower SIL – is mandatory at every download.</p>
<i>ControllerName</i>	<ul style="list-style-type: none"> • Connected Applications • Connected Libraries • Hardware Types • Hardware • Tasks • Access Variables • Main Source Units 	All items are mandatory
Linked Communication Variables	<ul style="list-style-type: none"> • Downloaded Controllers • System Controllers 	All items are mandatory

Information about execution order and linked communication variables will be part of the report, provided that a compilation has been performed.

Source code for protected types will not be displayed in the report. In the report, a protected type is indicated by a padlock icon . If the protected type is part of a library, it is possible to override the protection by entering the password.

To print the source code for the whole project, select **File > Print**. To print the source code for selected parts of the project, navigate the tree to the item you want to print, right-click the item and select **Print Source Code**. Alternatively, you can select **File > Print**, and select print range **Selection** in the Print dialog.



The source code report has a filter function to increase the readability of the source code for Function Block Diagrams and Control Modules. This filter is by default turned on (select **Tools > Filter**).



You can generate a source code report without compilation or download. See [Source Code Report Generated for Project in Control Builder](#) on page 249.

You can also generate a source code report for the project in the controller. See [Source Code Report Generated for Project in Controller](#) on page 249.

Reports Generated at Download

Difference Report and Source Code Report Generated at Download

For a description of the difference report and source code report generated when you perform a download of a project from the Control Builder to the controller, see [Difference Report](#) on page 237 and [Source Code Report](#) on page 245.



If a High Integrity controller is used, it is not possible to disable the Difference Report and start value analysis functions.

Source Code Report Generated for Project in Control Builder

To generate a source code report for the project in the Control Builder, without performing any compilation or download, select **Tools > Source Code Report**.

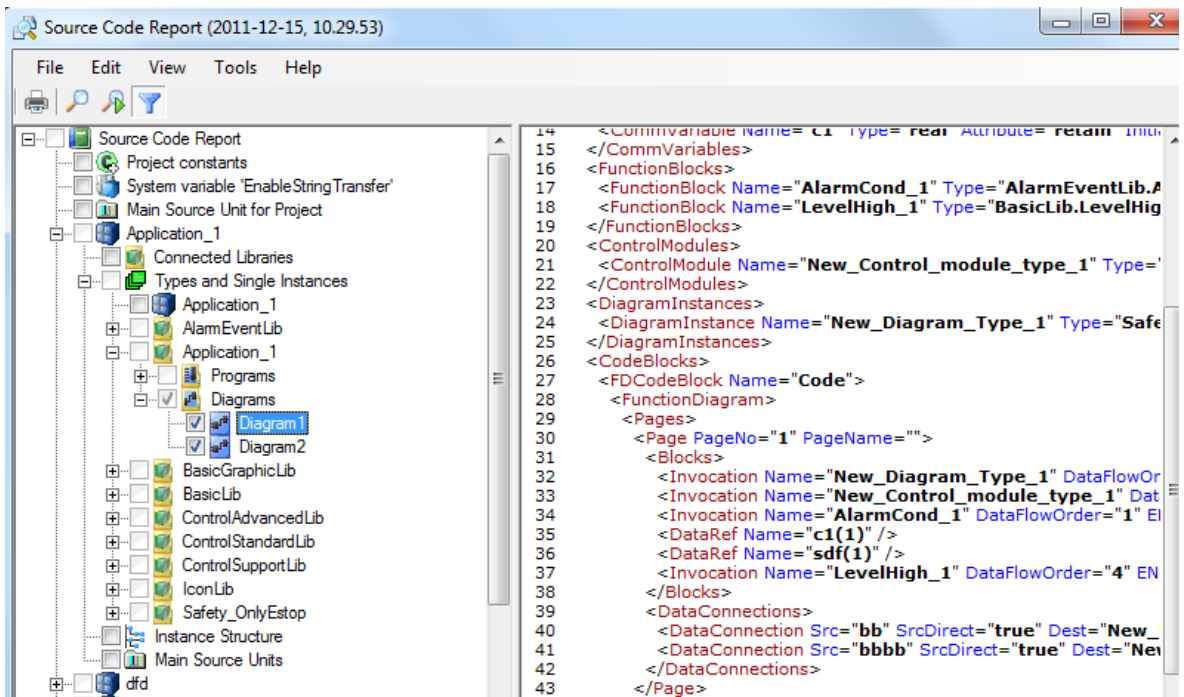


Figure 109. Source code report generated without prior compilation

Source Code Report Generated for Project in Controller

A source code report for the project running in the controller can be generated provided that:

- A successful download to the controller, with difference report enabled, has been performed.
- The project in the Project Explorer is the same as the project in the controller.

To generate a source code report for the project in the controller, right-click the controller in the Project Explorer and select **Remote System**, and then click **Show Downloaded Items**. In the Downloaded Items dialog, click **Source Code Report**.

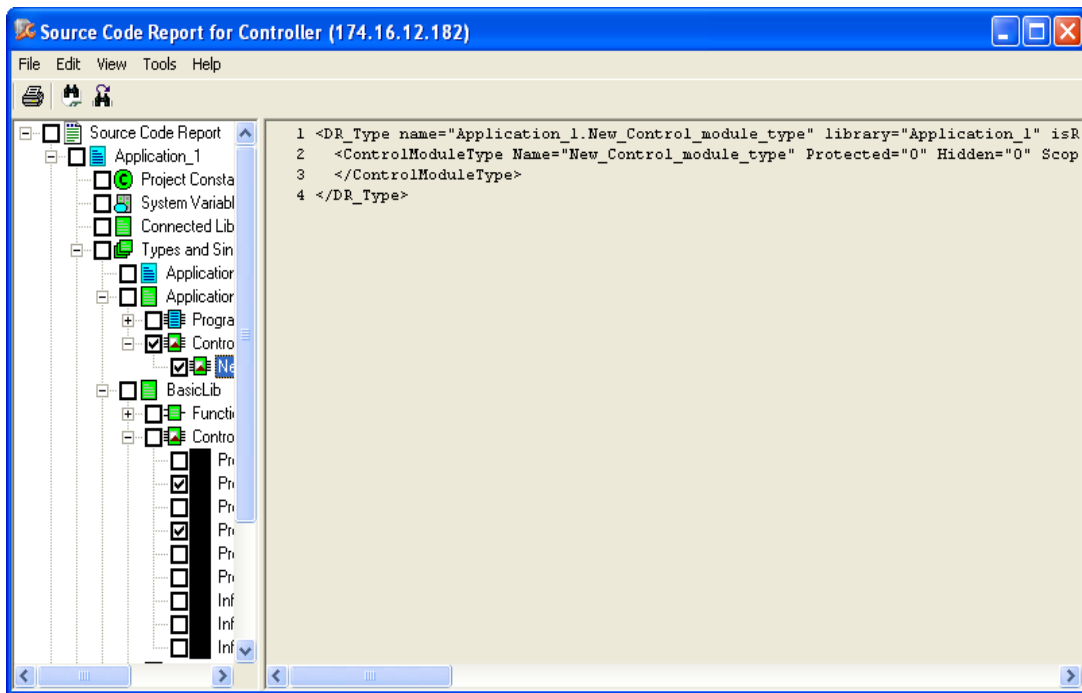


Figure 110. Source code report generated for project in controller.

Portability Verification

This menu is located under the menu option **Tools > Verify Portability** in the Project Explorer. This functionality verifies that the source code doesn't contain any characters with an ASCII value above 127. If a project, containing characters with ASCII values above 127, is moved between computers with different local system settings it may result in errors when the Control Builder project is loaded.

Performance Management

The compiler statistics is a separate tool accessible from the tools menu in the Control Builder as shown in [Figure 111](#). When it is started, the currently opened project in the Control Builder is compiled, and the collected information is saved in XML format in the 'Results' subfolder of the working folder, which is presented in a separate dialog, see [Figure 112](#). Only information about compilable applications can be gathered by the tool. If a project contains applications with errors, only statistics of the correct applications is presented. In this case the statistics presented is taken from a part of the project. One file is generated per application and it replaces old ones if it previously exists for that application. The tool can also be started from a Control Builder with no project loaded. In that case only information gathered from previously generated files is displayed.

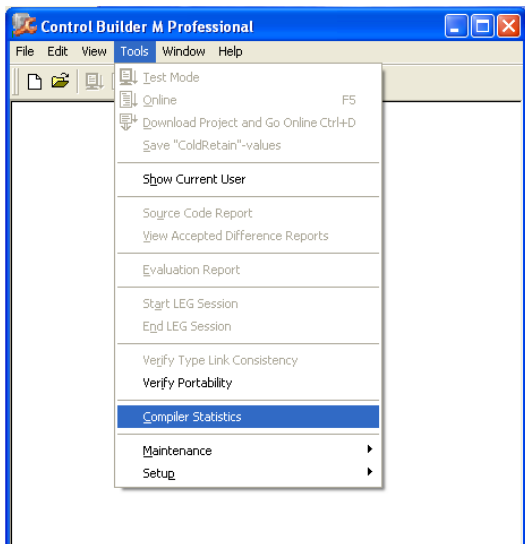


Figure 111. Compiler Statistics Tool

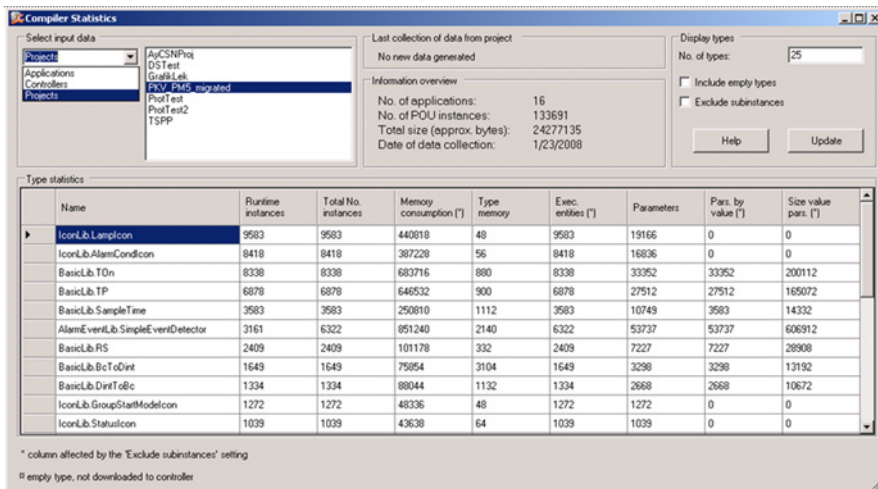


Figure 112. Compiler Statistics dialog

The tool presents the following information:

- The number of runtime instances of a type.
- The number of sub-instances of a type and total number of instances caused by the type.
- The memory cost of one instance of the type, both including sub-instances and without sub-instances.
- The total memory cost for all instances of the type. Both including sub-instances and without sub-instances.
- The number of code blocks of the type.
- The number of execution entities (Code calls) for all instances of the type, both including sub-instances and without sub-instances.
- The number of parameters passed by value of a type and the number of bytes passed by value.
- As above for all instances of the type, both including sub-instances and without sub-instances.

Project Documentation

Project Documentation function generates documentation of libraries and applications (in offline mode) for all the items in a selected project explorer folder. This function can be used to make a reference book for a library, an application, a controller, or a single object in a folder.

The project documentation function provides you with filter options while documenting your control project. The filter helps you specify parts of the control project and keeping the document size to a minimum. All documentation is produced as Microsoft Word documents as default, hence Microsoft Office must be installed.



All project documentation will be connected to a standard template. But you can create templates of your own for the documentation.

A complete overview of a library, an application, a controller, or an object in these folders can be exported to a file for printout from Project Explorer. However, it is not possible to select a folder at the root level, for example the Libraries object

folder. As an example, it is possible to filter out all ColdRetain variables and Parameters in an application.

If the project documentation function is used in Online mode, the cold retain values can also be obtained.

Printing Project Documentation

To print documentation, in Project Explorer:

1. Right-click any object in the tree view and select **Documentation**. The Documentation dialog box opens.
2. Click **More** to filter information. The Editor Properties dialog box opens.

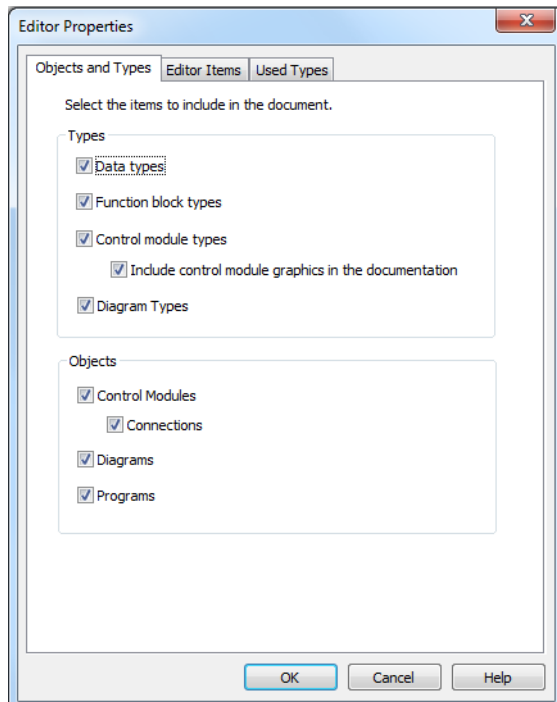


Figure 113. Editor Properties dialog for filter options.

The Editor Properties dialog box contains three main areas, which are represented by tabs in the dialog, see [Figure 113](#).

- Objects and Types,

- Editor Items,
- Used Types.

Objects and Types

This is the start level for filtering the contents of your application or library. As you can see, all options have been selected by default. You adjust the filter setting by exclude an option.

Editor Items

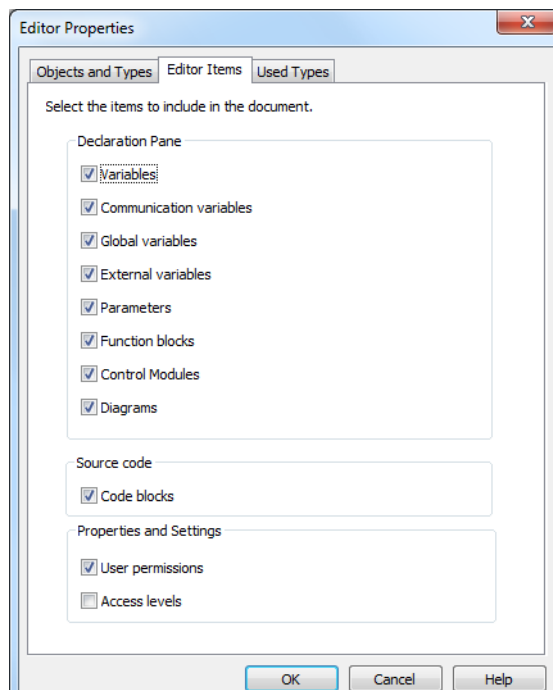


Figure 114. Editor Items tab for selecting items inside filtered types and objects.

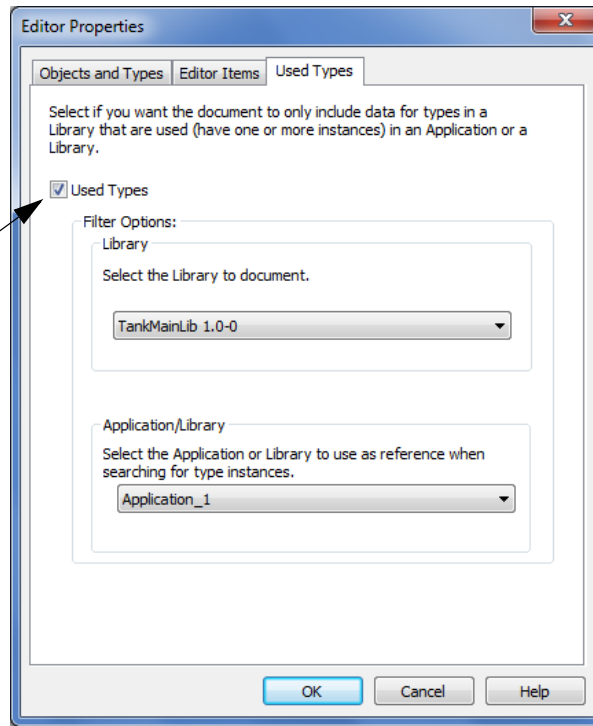
After adjusting the filter settings for types and objects, another filtering can be done per item. You can now specify which items to include/exclude for the previous selected types and objects. The items are grouped under Declaration Pane, Source code and Properties and Settings. All items are set by default, except the Access

Level option, (see [Figure 114](#)).



Access Levels are used for controlling access to online resources in SIL applications.

Used Types



Used Types must be checked.

Figure 115. Used Types dialog for printing used types only.

This filtering option selects types in a library that has an object (instance) in an application or inside another library. The resulting documentation from this dialog will only include the information for those types that have been matched as a reference in the selected application or library (see the drop-down menus in [Figure 115](#)).



In order to select a library or an application/library reference from the drop-down menus, you must first check the Used Types check box.

Section 2 Alarm and Event Handling

Introduction

An important part of an automation system is to be able to supervise and interact with the system. For this to be possible, information about the status of the supervised processes must be made available to the operator. Both the operator and the controllers need to be able to interact with the process.

This requires that information is transferred to and from the operator interface, in the form of commands, alarms, and events.

Alarms and events are generated in three ways:

- by using objects based on library types containing alarm and event functions,
- by using objects especially made for alarm and event handling (based on the types in the Alarm and Event library),
- by hardware units throughout the system (system alarms).

This section describes how to add alarm and event handling when there are no built-in functions for this. For information on how to configure alarm and event handling using objects that already contain alarm and event handling functions, refer to the *System 800xA Control AC 800M Binary and Analog Handling (3BSE035981*)*, and to online help for the object in question.



This chapter describes the alarm handling functions in the Alarm and Event library. Signal objects, process objects, and a number of control objects have built-in alarm functionality that is similar to the functions described in this section. For a description of built-in alarm functions, see the references above.

Alarms and Events

Alarms and events inform the operator of the status of processes and systems. An alarm represents a named state, also called an alarm condition (this is an OPC standard term). Events give information about changes that is needed to analyze various error situations. The OPC standard defines three kinds of events:

- Condition-related events, which are created when an alarm state changes.
- Simple events, which are created at occurrences like when a motor starts.
- Tracking-related events, which are created at occurrences like an operator action.

Alarms are usually presented to the operator in alarm lists, while events are presented in event lists. Alarms and events can also be handled by various parts of the system without the involvement of an operator, so that, for example, a process is stopped when a certain alarm goes on. Alarms and events, the functions can be used in SIL applications, but they are restricted to be used only for non-SIL purposes, for example indications and does not influence the critical loop. Any violation of this might corrupt the safety application and in SIL3 it could also lead to a safety shutdown.

Alarms and events are collected from controllers and other parts of the system, and transferred to subscribing OPC clients (operator interfaces) using an OPC server, see [Alarm and Event Communication](#) on page 289.



The behavior of an AC 800M High Integrity controller is in some cases different from the behavior of other controllers. Limitations that apply when running SIL applications in a High Integrity controller are described in the *System 800xA Safety AC 800M High Integrity Safety Manual (3BNP004865*)*.

Alarms and events are often logged, for use in trouble-shooting and when tracing the origins of an error, see [Section 5, Maintenance and Trouble-Shooting](#).

There are two main types of alarms and events:

- Process alarms and events are generated by changes in the alarm condition of a monitored process signal, see [Process Alarm and Event Generation](#) on page 259.
- System alarms and events are generated by a change in the status of the system itself, for example by a hardware failure or by the application via function

block (SystemAlarmCond). See [Detection of Simple Events](#) on page 269 and [System Alarm and Event Generation](#) on page 282.

Alarm and event handling also requires clock synchronization, in order for time stamps to be reliable when trying to analyze a sequence of events. See [Time Stamps](#) on page 286 and [Sequence of Events \(SOE\)](#) on page 274.

All alarms and events follow the OPC Alarm and Event specification.

Alarm and Event Library

The Alarm and Event library contains function blocks and control modules for:

- Creating alarms and events when a monitored signal of type bool changes,
- Creating simple events with user-defined data, for use in, for example, batch applications,
- Printing alarms and events.

Additional Information

For examples of how to use components from the Alarm and Event library, see [Alarm Examples](#) on page 297. For details on how to use alarm and event functions, see [Alarm and Event Functions](#) on page 316. This sub-section also describes how to set up printers and print queues.

For a complete list of all objects in the Alarm and Event library, see the manual *System 800xA Control AC 800M Binary and Analog Handling (3BSE035981*)*. For information on which alarm and event types that can be used in SIL applications, see online help for the Alarm and Event library or the reference above.

Process Alarm and Event Generation

Process alarms and events can be generated using a number of objects based on types in the Alarm and Event library.

- The function block types AlarmCond and AlarmCondBasic, as well as the control module types AlarmCondM and AlarmCondBasicM, can be used to generate alarms and events each time there is a change in a monitored signal (of

type bool). See [Process Alarms and Events](#) on page 260.



The function block type AlarmCondBasic and the control module type AlarmCondBasicM are versions of AlarmCond and AlarmCondM, which consume less memory. These types do not allow inverting the monitored signal and they support internal time stamps only.

- The function block type SimpleEventDetector can be used to generate a simple event whenever a monitored signal of type bool changes. See [Detection of Simple Events](#) on page 269.
- The function block type DataToSimpleEvent can be used to create a simple event and add user-defined data to it. [Detection of Simple Events](#) on page 269.

There are also system generated alarms and events, see [System Alarm and Event Generation](#) on page 282.

Process Alarms and Events

Alarm condition-driven alarms and events are created when the monitored signal changes, that is, when an alarm condition is fulfilled. This monitored signal must be of type bool and is typically taken from another function block or module in the system, or from an external device. The alarm condition function blocks and control modules are state machines, which change from one state to another following a set of configurable rules, whenever the monitored signal changes. This is defined as a change in the *alarm condition*. Each time an alarm condition changes, an event is created as well.



All alarm condition objects can be used in time-critical tasks and also most of them in SIL certified applications.

AlarmCond and AlarmCondM

The two basic types for creating alarm conditions are the function block type AlarmCond and the control module type AlarmCondM. The principle behind the two is the same. Through parameters, it is possible to connect to the monitored signal, add information to the alarm, provide other objects with status information, and to control the behavior of the alarm condition. In [Figure 116](#), the function block type AlarmCond is used to illustrate the function of the different parameters.

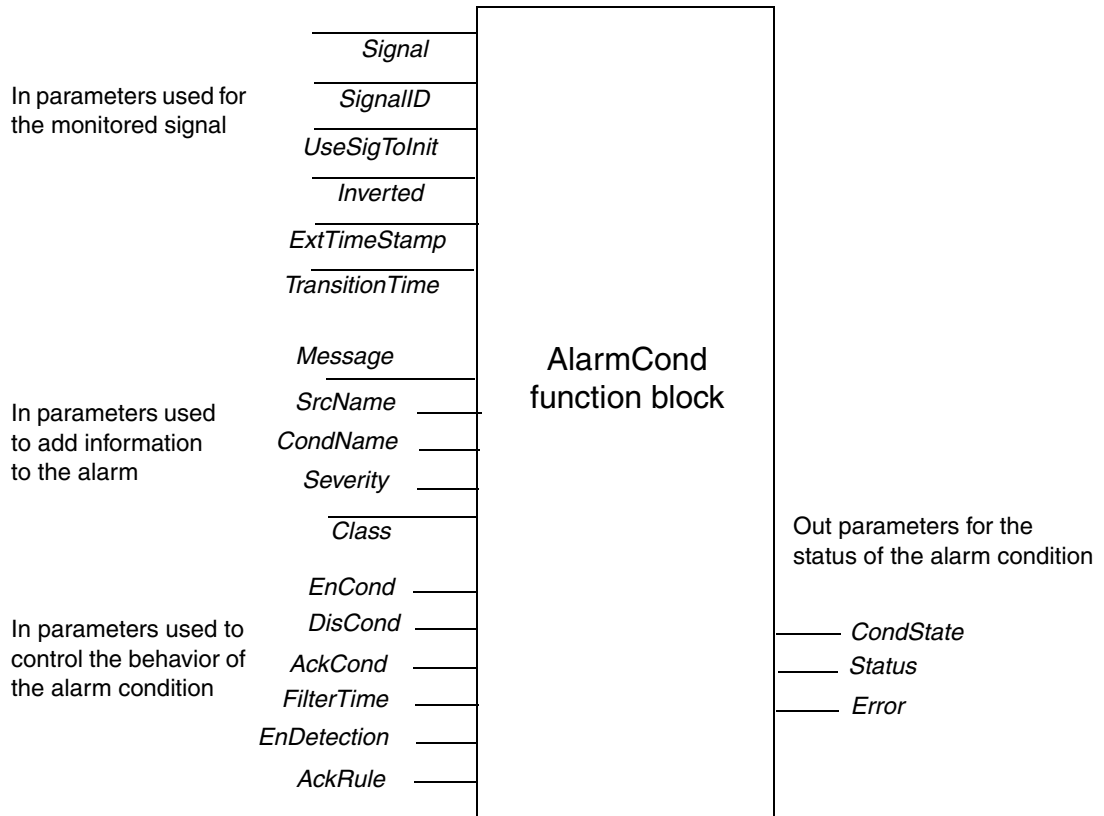


Figure 116. The function block AlarmCond.



If you change the value of an *Edit* parameter, this change will not take effect until after a warm or cold download.

The following alarm condition parameters are *Edit* parameters:

- *ExtTimeStamp*,
- *SignalID*,
- *UseSigToInit*,
- *SrcName*,
- *CondName*,
- *Inverted*,
- *AckRule*.

The Description field in the parameter editor starts with EDIT if the parameter is an *Edit* parameter.

The control module type AlarmCondM has similar functions and uses the same parameters as the AlarmCond function block type.



For more information on parameters and their possible values, also see online help and the Description column in the parameter editor.

Alarm Condition Types with Reduced Functionality

In applications where it is necessary to minimize memory consumption, the function block type AlarmCondBasic and the control module type AlarmCondBasicM offer an alternative to AlarmCond and AlarmCondM.

Basically, they are the same as their counterparts AlarmCond and AlarmCondM, with the following differences:

- They consume less memory.
- They always use acknowledgement rule number 1 (*AckRule*=1).
- It is not possible to invert the in signal, that is, the *Inverted* parameter cannot be used.
- External time stamps cannot be used, that is, the parameters *ExtTimeStamp* and *SignalID* are not used.
- Remote time stamps cannot be used, since the parameter *TransitionTime* cannot be used.

Select Signal to Monitor

The monitored signal can be internal (that is, reside in the controller), or external (that is, reside outside the controller).

Which type of signal that is monitored is indicated by the parameter *ExtTimeStamp*. If this parameter is True, the external signal indicated by the hardware address in the parameter *SignalID* is monitored. If *ExtTimeStamp* is false, the parameter *Signal* is used to connect to the monitored signal.

The parameter *Inverted* can be used to invert the in signal (True=invert signal).

UseSigToInit is used to indicate from where the initial value of the signal should be taken (the state machine needs a start value). This parameter is only relevant when the monitored signal is external. When *UseSigToInit* is True, *Signal* is used to get an initial value.

Control the Behavior of the Alarm Condition

The following parameters can be used to control the behavior of an alarm condition:

- *AckRule* determines which acknowledgement rule is used. The acknowledgement rule decides the behavior of the alarm condition when an alarm has been created. This parameter is an EDIT parameter (that is, it is used for configuration purposes only, and cannot be changed without a restart) and it cannot be changed from the code.
- *FilterTime* determines how long the signal must deviate before a change is considered to have taken place. The filter time should be set so that glitches do not cause an alarm.
- *TransitionTime* determines the time of the event occurrence when the Signal change. If the value is equal the default value (the time) will be read inside this FB instead
- *EnDetection* enables detection when True. When this parameter becomes False, the alarm condition goes to an inactive state and the signal is no longer monitored. By setting this parameter to False, you will stop detection of new alarms and leave existing alarms unacknowledged.
- *AckCond* is used to acknowledge an alarm (True = acknowledge). It is normally used to acknowledge alarms from simple devices such as push buttons.

- *DisCond* disables the alarm condition when True.
- *EnCond* enables the alarm condition when True.

How the condition state changes when an alarm is acknowledged depends on the value of the acknowledgement rule (*AckRule*) parameter. This parameter is available in the AlarmCond and AlarmCondBasic function blocks, and in the AlarmCondM and AlarmCondBasicM control modules.



The *AckRule* parameter is normally set to 1 (normal). It cannot be changed online.

There are five acknowledgement rules:

- *AckRule* = 1, “normal handling”, alarms must be acknowledged and inactive before the “normal” state is resumed,
- *AckRule* = 2, alarms need no acknowledgement,
- *AckRule* = 3, alarms return to “normal” state on acknowledgement,
- *AckRule* = 4, not used (reserved for future use),
- *AckRule* = 5, alarms return to “normal” state when a sum system alarm is acknowledged and returns to its normal state.

For more information about the different acknowledgement rules, see [Acknowledgement Rules – State Diagrams](#) on page 317.

Alarm and Event Information

There are a number of parameters for adding information to alarms and events:

- *Message* can be used to add a textual description of the alarm condition, for example, “temperature low”.
- *SrcName* identifies the alarm source, for example, “Motor101”.
- *CondName* identifies the alarm condition, for example, “Level_High”.
- *Severity* indicates the degree of severity, where 1 is the least severe, and 1000 is the most severe level. This parameter is very useful when filtering alarms and events.
- *Class* can be used to classify the alarm (1-9999). This parameter is also useful when filtering events,

This information can be displayed in the operator interface and written to various logs. It can also be used to sort and filter alarms and events.

Since the source name and the condition name identify the alarm, the combination of the two must be unique within a controller. Any attempt to define an alarm condition that results in a non-unique combination of source name and condition name will result in an error (the *Error* parameter will become True). Also, a simple event is generated.

If an OPC server detects a non-unique alarm (that is, two controllers have the same combination of source name and condition name), a system simple event is generated.

There are two alternatives for indicating the source of an alarm or event:

- Leave the *SrcName* parameter empty. The *Name* parameter of the alarm owner (see [Alarm Owner Concept](#) on page 268) will be used as the source name.



For a program or application to have a source name, you need to create a variable called *Name* in the program or application. If the *SrcName* parameter is left empty and the alarm owner is a program or application, the value of the *Name* variable will be used as the source name.

- Set the *SrcName* parameter to whatever source name you want to use.



All alarms belonging to the same alarm owner must have the same source name.

The condition name is normally the name of the alarm condition function block or control module instance, for example *Level_High*, but could also be set via the *CondName* parameter.



Condition names are case sensitive, that is, *Level_High* is not the same as *LEVEL_HIGH*.

The same condition names should be used throughout the whole project, since it is important that the operator has a limited set of condition names to deal with. Using condition names in a consistent and structured manner also makes it easier to understand the process.



For detailed information about source name and condition name restrictions and syntax, see online help for the Alarm and Event library. For information on NLS handling for alarms and events, see [Translation – NLS Handling of Strings](#) on page 296.

The class parameter (*Class*) can be used to classify all alarms.



The default class is 9950 for all system alarms and system events. All other numbers can be used as required. Possible values are 1-9999. The default value can be changed by changing the CPU setting *AE System AE class*.

Status Information

There are three parameters that can be used to retrieve status information for an alarm condition:

- *CondState* indicates the state of the alarm condition (0-6, see below).
- *Error* indicates an error in the alarm condition.
- *Status* gives the status code from the latest execution.



If a parameter is outside its defined range, the *Status* parameter will take a negative value or the value 703.

Alarm conditions are state machines, which change from one state to another following fixed rules. The most important reason for an alarm condition to change is a change in a monitored signal. The alarm condition (indicated by the parameter *CondState*) also changes if:

- an alarm is acknowledged,
- an alarm is disabled,
- an alarm is enabled,
- auto-disable occurs.

The condition state (*CondState*) parameter indicates the state of an alarm. An alarm can be in one of seven states:

Integer value	State
0	Alarm condition not defined
1	Disabled
2	Enabled, Inactive, Acked - Idle
3	Enabled, Inactive, Unacked
4	Enabled, Active, Acked
5	Enabled, Active, Unacked
6	Enabled, AutoDisabled, Unacked

The *CondState* parameter can be used to pass the state of an alarm to other parts of the software.



To see the state of all alarm conditions for a certain object in Project Explorer, right-click the object and select **Alarm Conditions** from the context menu.

Autodisable

AC 800M controllers have a CPU parameter called *AE Limit auto disable*. This setting controls the number of times an alarm can go on and off, without being acknowledged. When the limit is reached, the alarm condition is automatically disabled, and the state AutoDisabled is entered. The default setting is 3, and the maximum setting is 127. If *AE Limit auto disable* is set to 0, autodisable is turned off and alarms can be activated an unlimited number of times.



An alarm that is in AutoDisabled state does not send any event (even though the alarm condition changes), until it is acknowledged. See [Acknowledgement Rule 1](#) on page 317.

Alarm Owner Concept

The alarm owner concept is important, since it is the key to manipulating the source of an alarm. Not all objects in the Project Explorer tree hierarchy are alarm owners. For an object (for example, a tank object) to be an alarm owner, it must fulfill three criterias:

1. It must have the attribute Alarm Owner set to True.
2. It has to be the last link in an unbroken chain of alarm owners, all the way from the program or application, down to this particular object. For an illustration of the concept, see [Alarm Owner Examples](#) on page 304.
3. It must have Aspect Object set to true.
4. It must not be a sub object to an object with the attributes hidden or protected.



The attribute "Sub Objects visible in PPA" will not affect the Alarm ownership.

If an object is not an alarm owner, or the alarm owner chain is broken, the system looks further up in the hierarchy, until it finds an object on a higher level that is directly above the origin of the alarm or event, and fulfills the above criteria.

This is the point of the alarm owner concept. By not setting the Alarm Owner attribute for low-level objects, alarms and events can be connected to an object on a level higher than their true origin. If no alarm owner is found, the program or application itself becomes the alarm owner. The following objects are always alarm owners:

- Applications,
- Programs.



The Name parameter of the alarm owner or the Name variable corresponding to the name of the alarm owner must be initialized before the alarm condition changes (triggering an alarm).

Each object that is an alarm owner creates three aspects, which can be viewed from both the Object Type Structure and the Control Structure:

- The Control Alarm Event aspect lists all alarm conditions associated with the object.
- The Alarm List aspect presents alarms associated with the object during operation.

- The Event List aspect is used to present an event list.
- If you want to, fshow alarms for all objects, you can override the above aspects.



Detection of Simple Events

A simple event detector generates a simple event each time there is a change in the monitored signal. A simple event detector can be implemented by means of the function block type SimpleEventDetector.

SimpleEventDetector can be used with internal, external or remote time stamps. This function block type is connected to the monitored signal exactly the same way as the function block type AlarmCond, that is, using the parameters *Signal*, *SignalID* and *UseSigToInit*. See [Select Signal to Monitor](#) on page 263. It is also possible to set the filter time (via a *FilterTime* parameter).



For SimpleEventDetector, the following applies:
If *ExtTimeStamp* is True, *FilterTime* is not used.

The function block DataToSimpleEvent can be used to add data to a simple event. See [Simple Events](#) on page 286.

For more information on how to configure these function blocks, see alarm and event online help.

Built-in Alarm and Event Handling in Other Libraries

This section deals with alarm and event handling based on the Alarm and Event library. However, alarm and event functions are built in to a number of other types in the standard libraries that are delivered with the 800xA system.

This sub-section gives a short introduction to signal objects and to the built-in alarm and event functions of process objects and control loops. It also describes the inhibit and disable functions for these objects, since they are relevant to the interaction with the types in the Alarm and Event library.

Alarm and Event Handling Using Signal Objects

The Signal Library contains types that can be used to create representations of objects with an input or output signal, for example a temperature sensor. By using a signal object, you can go to manual mode and set the value of the signal, as well as supervise the signal and generate alarms when the signal deviates.



Never use types from the Signal Libraries to represent *all* I/O channels and if used, types from SignalBasicLib should be taken. This will consume a lot of memory and will result in poor performance. Use signal objects when there is a real need to control and monitor an I/O signal. Signal objects normally represent an object with a single signal.

For more information about the Signal Libraries, see online help and the manual *System 800xA Control AC 800M Binary and Analog Handling (3BSE035981*)*.

Alarm and Event Handling in Control Loops and Process Objects

Alarm and event handling is built into a number of library types, such as control loops and process objects. These alarms and events are handled the same way as other process alarms and events.

Alarms and events can be generated directly by those objects, each time the alarm condition is fulfilled, or the object can generate a bool signal that can be connected to an alarm condition object.



For a description of how to configure built-in alarm handling for various library types, see online help for the type in question, and the *System 800xA Control AC 800M Binary and Analog Handling (3BSE035981*)*.

Inhibit and Disable Alarms and Events

Sometimes there is a need for temporarily suspending alarm and event generation. This can be done for all objects with built-in alarm handling:

- **Disable** – the alarm condition is disabled, no alarms and events are generated, nothing is sent, and no control action is taken (that is, the system does not act upon the alarm condition).



Normally, the control action will be a boolean signal that causes a certain reaction, for example, a signal that stops a motor. However, a control action could also cause a more complex series of actions.

- **Inhibit** – the control action itself is inhibited (that is, the system does not act upon this alarm or event), while alarms and events are still presented to the operator in the operator interface.



Inhibit is only available in the types listed under [Inhibit Parameters](#) on page 272.

Alarms and events can be disabled from the faceplate and from alarm list, as well as from the application, via interaction parameters. [Figure 117](#) illustrates the difference between inhibiting and disabling an alarm.



In a SIL application, alarms cannot be enabled or disabled via the MMS event service. However, alarms can be disabled or enabled from the IEC-61131-3 code.

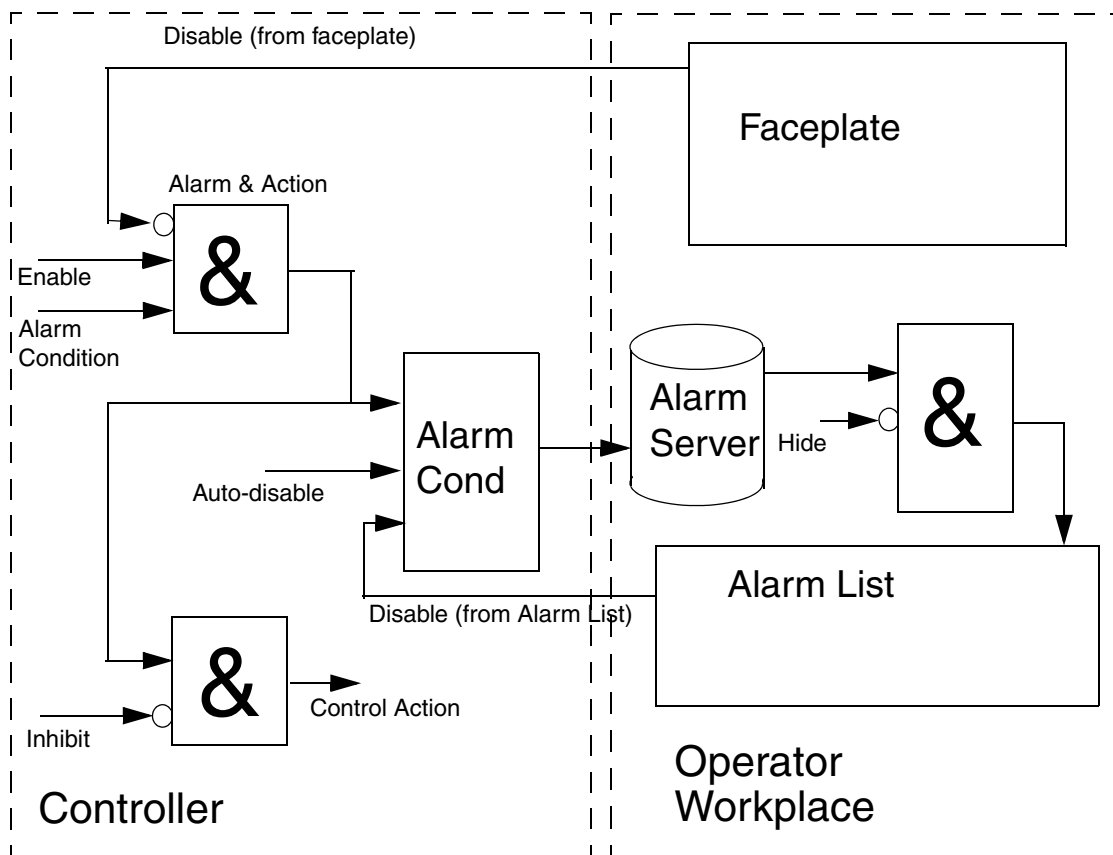


Figure 117. Inhibit and disable functions in alarm handling for AC 800M (for a list of objects with inhibit functionality, see [Inhibit Parameters](#) on page 272). Event handling works in a similar way.

In the above figure, the alarm can be disabled from the faceplate and from the alarm list. It can also be disabled if the auto-disable function is triggered.



Disable from the alarm list only disables the alarm itself, while *Disable* from the faceplate disables both the alarm and the control action connected with it.

When *Inhibit* is set, the alarm still exists and can be seen in logs, face plates and alarm lists. It is only the control action that is inhibited.

Hide is set from the operator interface, see operator workplace documentation.



For an example of how to use the inhibit function, see [Alarm Examples](#) on page 297.

Inhibit Parameters

The inhibit function is present in the following standard library types.

- Signal library
 - SignalInReal
 - SignalReal
 - SignalInBool
 - SignalBool
 - SDLevelM
 - SignalBoolCalcInM
 - SignalInBoolM
 - SignalInRealM
 - SignalReadCalcInM
 - SignalSimpleInRealM
- Standard Control library
 - Level6CC
 - Level4CC
 - Level2CC
- Supervision library
 - DetectorBool
 - Detector1Real
 - Detector2Real
 - DetectorRemote
 - DetectorLoopMonitorReal
 - OutputBool

- SupervisionOverview (no disable function)
- OutputOrder (no disable function)
- Supervision Basic library
 - SDBool
 - SDInBool
 - SDInReal
 - SDLevel
 - SDOutBool
 - SDReal
 - SDValve
 - InfoAlarmSDInReal
- Fire&Gas library
 - FGOutputOrder (no disable function)
 - CO2 (no disable function)

In these types, control actions are inhibited by setting a parameter *InhXAct*, where *X* stands for the name of the condition, for example *InhGTHAct* (where GTH stands for Greater Than High).

There are also parameters for indicating if the alarm condition (event generation) has been inhibited or not.

Disable/Enable Parameters

The disable function is available in all types that contain built-in alarm handling. An alarm condition is disabled by setting the *EnableY* parameter to False, where *Y* stands for the name of the condition, for example *EnableGTH* (where GTH stands for Greater Than High).

There are also parameters for indicating if the alarm condition has been disabled or not.



There are additional parameters that affect the behavior of built-in alarm conditions, for example *AEConfigX*. For more information on parameters, see online help for the object in question (select and press F1).

External Time Stamps (S800 I/O)



A special form of external time stamp is created by external units with Sequence-of-Event (SOE) support, such as DI831. A low level event is then time-stamped by the I/O unit and sent to the controller to be dealt with. This triggers alarms or simple events in the controller. The change of status is time-stamped with the low level event time stamp.

Sequence of Events (SOE)

Some I/O modules add a low-level time stamp to an alarm or event when it detects a change in a signal. Instead of using the time stamp created by the controller when it detects a change in the monitored signal (that is, when the task is executed), the controller simply adds the time stamp created by the I/O module. In this way, the time stamp shows when the change actually occurred, instead of when it was detected by the controller.

For this to work, the I/O module will have to support Sequence of Events (SOE). SOE is currently supported on ModuleBus and PROFINET IO only. For information on enabling/disabling and configuring SOE, see online help for S800 I/O.

External Time Stamps IEC 61850

Sequence of Events (SOE) for IEC 61850

CI868 module supports Sequence of Events for incoming Circuit Breaker position status containing IED Time Stamps (External Time Stamp) reported from other IED.

The IEDs associated with the Circuit Breaker IEDs adds a time stamp value along with status value and updated in RCB dataset whenever a change in Circuit Breaker position is detected.

CI868 module captures this status and external time stamp value assigned as IEC 61131 - 3 variable in CSWI or XCBR LN objects and generates respective External Events.

These External Events can be used as input for AlarmCond and SimpleEventDetector function block in IEC 61131 -3 application to generate Alarm and Events.

In this way the Alarm and Events generated from External Events will have the following attribute:

- Alarm and Event with Time stamp of IED instead of AC 800M Controller.
- Alarm and Events with source objects mapped to the process Object or Conducting Equipment instead of CSWI or XCBR hardware object library in Control Builder tree.
- Alarm and Events generated as process Alarm and process Events instead of System Alarm and System Events, thereby allowing further categorization of process Alarm and Event in Alarm and Event list.

For more information on External Events refer to the *AC 800M IEC 61850 Configuration for CI868 (9ARD171385*)* manual.

External Time Stamps (PROFINET IO)

Sequence of Events (SOE) for PROFINET IO

Time stamped events are passed by PROFINET IO and CI871 through the controller and are indicated in the AC 800M OPC Server in the Engineering Workplace. The time stamping is done by the PNIO device. The PROFINET IO SOE is supported by use of the ABB SOE profile.

The following are the definitions and functions of ABB SOE Profile:

1. Alarms from the PNIO device are converted into an External Event. These External Events transferred through the AC 800M OPC-Server are indicated in the EventList with their corresponding source address.
2. The external event can be picked up from the IEC-61131 Application by a Function block like alarm condition and converted to a process alarm.
3. The time synchronization of PNIO device is done externally and not by the CI871. It is the responsibility of the PNIO devices to get a time synchronization managed (through access to the central time master in the system). The PNIO device defines the information to be time stamped.

4. The ABB SOE profile is handled as a process alarm on PROFINET IO with a vendor specific User Structure Identifier (USI).
5. Once the SOE alarm is acknowledged (to ensure that it is not lost). The PNIO device deletes the alarm only after receiving the acknowledgement from the controller. The controller sends the acknowledgement after storing the alarm in the non-volatile memory.



It is recommended to configure Function blocks as Alarm condition for process signals only where the process values can be used as initial value in case of restart behavior. Otherwise alarms can get frozen.

External Time Stamps (INSUM)

Creating an Application that Handles INSUM Alarms

All INSUM devices (MCU, Circuit Breaker) have supervision functions that can report alarms. The different device types supervise and report specific alarm types. The alarms are reported in specific Network Variables.

MCUs report the alarms in the Network Variable NVAlarmReport.

The user can decide if there should be a summary entry that tells that there are some alarms (one or more) in the device. It is possible to have a separate summary alarm for warnings and a separate alarm for trips.

This subsection discusses both methods, receiving INSUM alarms in the application program, and generating alarm to the alarm lists. The user can decide to use either methods or just one of them. For more information refer to *System 800xA Control AC 800M Binary and Analog Handling (3BSE035981*)* manual.

Receiving INSUM Alarms in the Application

To receive alarms in the application program the INSUMReceive function block is used in the same way as when receiving other input network variables from an INSUM device, choose the correct NVindex and data type. The data type should in this case be NVAlarmReport (see also the MCUAlarmTrips/WarningsStructs regarding how to interpret the bits).

The time stamp set by the INSUM device in the alarm variable is presented in the two time fields of the NVAlarmReport. This time information is only correct if the

clock in the INSUM device is synchronized. The system software does not fill in these fields if the time stamp received from the INSUM device is incorrect. (See below).

Generating Alarms for Alarm Lists

The controller system software generates alarms for the alarm and event lists in the system, based on the updates of the INSUM alarm information if the parameter *Generate Alarms* on the device is set to *Enabled* or *Enabled Trip/Warning* or *Enabled Detailed*.

If the time stamp received from the INSUM device is correct (a valid time) this time stamp is used for the generated alarm message. If it is not, the system software tags the generated alarm message with the current controller time.



If the parameter *Generate Alarms* is set to disabled, alarm information can anyway be sent to the alarm and event lists by the application. This can be done by creating an *AlarmCond* function block and to connect information received from an INSUM device to the parameter *Signal* and to set *External Time Stamp = FALSE*.

In this case, the alarm messages are time stamped in the controller. If this time accuracy is sufficient, this method is probably to be recommended because it is easier to configure. No System Clock is needed in the INSUM system. If you let the system software generate the alarms it can use the time stamp given by the INSUM devices. If the INSUM System Clock is used this is a much more accurate time stamp.

Summary Alarms, One Alarm Object Per Device

Generate Alarms = Enabled means that the system software internally (without needing *INSUMReceive*) creates a subscription of the alarm variable from the INSUM device. When this variable is updated from the INSUM system, the system software evaluates the content.

If a bit (one or more) which is classified as an alarm (e.g. not the bit "Started1") is set and no such bit previously was set, the system software generates one alarm message.

If an alarm update is received with the change that no alarm classified bits are set any more, the system software generates the *alarm-off* message.

Summary Alarms, One Alarm Object For Warnings and One for Trips

Generate Alarms = Enabled Trip/Warning. The difference compared to the handling for *Enabled* is that the system software generates one specific alarm message when a warning bit is set and another alarm message when a trip bits are set.

This means that there will be one alarm message for the first warning and one for the first trip. To use this setting two AlarmCond blocks should be created for each INSUM device, one for the warnings and one for the trips. If an alarm update is received with the change that no warning bits are set there will be an *alarm off* message for the warnings. The same applies for the trip bits.

Detailed Alarms

Generate Alarms = Enabled Detailed. The difference compared to the handling for *Enabled* (see [Summary Alarms, One Alarm Object Per Device](#) on page 277) is that for each alarm classified bit which is set (and previously was not set) the system software generates one separate alarm message. If an alarm update is received with the change that an alarm classified bit that previously was set now is reset, the system software generates the *alarm off* message for that bit.



Using *Enabled Detailed* means that one *AlarmCond* block should be created for each alarm type that the INSUM device sends. For a large INSUM configuration where more than just a few alarm types per device should be supervised this easily leads to a very large number of AlarmCond blocks.

Creating AlarmCond Blocks for Generated Alarms

The function block AlarmCond should be used to get descriptive messages in the event and alarm list and get an association with an alarm object. AlarmCond is associated with the alarm messages that the system generates by setting *ExternalTimeStamp=TRUE* and to identify the alarm object with the parameter *SignalId*.

Alarm Generation = Enabled

The SignalId should be a string that specifies the hardware position for the INSUM device. This is done with the syntax *C.G.D*, where:

- C is the position of the CI857,
- G is the position of the INSUM Gateway and,
- D is the position of the INSUM device. The position numbers are separated by a dot '.'.

Example:

- The syntax *2.1.204* means the alarm for device #204 connected via Gateway #1 on CI857 #2.

Alarm Generation = Enabled Trip/Warning

The SignalId should be a string that in addition to the hardware position for the INSUM device, also specifies a trip or a word.

This is done with the syntaxes *C.G.D-T* or *C.G.D-W*, where:

- C, G and D as above,
- T represents Trips and W represents Warnings.

Examples:

- The syntax *2.1.204-W* means a warning for device #204 connected via Gateway #1 on CI857 #2.
- The Syntax *2.1.204-T* means a trip in device #204 connected via Gateway #1 on CI857 #2.

Alarm Generation = Enabled Detailed

The SignalId should be a string that, in addition to the hardware position for the INSUM device, also specifies the alarm word and bit within the word. This is done with the syntax *C.G.D-X/B*, where:

- C, G, and D as above, and,
- X is the word within NValarmRep (preceded by a dash "-"),
- B is the bit within the word.

There are four words with warnings called W0-W3 and four words with trips called T0-T3. The bits are numbered from 0 to 15. The word and the bit is separated by a slash '/'.

Example:

The syntax *2.1.204-W1/3* means the alarm bit 3 in word W1 in device #204 connected via Gateway #1 on CI857 #2.

Choose Alarm Handling Method for INSUM Alarms

This section contains some suggestions about choosing and handling INSUM alarms. Whether to send alarms to alarm list or not:

- If Alarms should be possible to view, but are not necessary to see in the Alarm lists:
 - *Set Generate Alarms = Disabled.*
 - Do not create any AlarmCond blocks.
- If the INSUM Alarms should be sent to the alarm list:
 - Use AlarmCond function blocks. See [INSUM Alarms in Alarm Lists](#) below.

INSUM Alarms in Alarm Lists

Time stamping:

- If local (in the INSUM devices) time stamping should be used:
 - Use a system clock in the INSUM system.
 - *Set Generate Alarms = Enabled, Enabled Trip/Warning, Enabled Detailed*
 - Use an AlarmCond block with *External Time Stamp = TRUE.*
- If it is sufficient with time stamping in the application in the controller:
 - *Set Generate Alarms = Disabled*
 - Use an AlarmCond block with *External Time Stamp = FALSE.*
 - Connect it to the variable with the INSUM device information to be supervised. The accuracy of this time stamping cannot be better than the cycle time of the application where the AlarmCond is executed.

Separation of alarms in the alarm list:

- If the timing between different alarms within a device must be possible to see in the alarm list than it is required to:
 - *Set Generate Alarms = Enabled Detailed.*
 - Use one AlarmCond per alarm type.

- If it is sufficient to be able to identify the device than it is possible to:
 - Set *Generate Alarms = Enabled*.
 - Use one AlarmCond per INSUM device.
- If it is sufficient to be able to identify the first warning and the first trip in a device than it is possible to:
 - Set *Generate Alarms = Enabled Trip/Warning*
 - Use two AlarmCond blocks per INSUM device.

Number of devices:

- If there are a lot of devices needing external time stamping than required for:
 - Use two (or one) AlarmCond per INSUM device.
 - Set *Generate Alarms = Enabled Trip/Warning* (or *Enabled*)
- If there are a few devices that need external time stamping than it is possible to:
 - Use one AlarmCond per alarm type.
 - Set *Generate Alarms = Enabled Detailed*

System Alarm and Event Generation

System alarms and system simple events that are generated in a controller are distributed to OPC alarm and event clients and locally connected printers, according to the current system configuration.

All system alarms available in a controller can be located by printing all alarms (use the PrintAlarms function block type and set the parameters to show the alarms you want to see). They can also be displayed by and interacted with applications, by means of the function block AttachSystemAlarm (this function block type retrieves the alarm condition state and some other information for an alarm condition). When units that are visible in Project Explorer (hardware units or program tasks) generate system alarms or system simple events, a warning icon is displayed on the corresponding unit.

System alarms and system simple events are used to draw attention to deviations from normal system behavior. All system alarms and system simple events can be sent to the OPC Alarm and Event Clients and even printed to the system log file, depending on the current system configuration.

Alarm Source name

The Alarm Source name functionality makes it easier to identify units in an alarm list. Its function can be accessed from the Controller aspect and is used to add an OPC Source Name aspect to all underlying hardware units and System Alarm and Event units.

For a hardware unit the name set in the OPC Source Name aspect will be a combination of the controller name and the unit path e.g. Controller_1-0.4.0.

For a system alarm and event unit, the name set in the OPC Source Name aspect will be the system alarm and event unit with the IP address replaced by the controller name.

From the Controller aspect.

1. Select the **System Alarm Info** tag in the Controller's aspect preview pane.
2. Click the 'Generated System Alarm Info' button.

The text in the Name field (in the OPC Source Name aspect) will be the name presented in the alarm list.

Controller Generated System Alarms and System Simple Events

Controller generated system alarms and system simple events are defined within the controller. A list of all defined system alarms and system simple events within an AC 800M controller can be found in [Appendix B, System Alarms and Events](#).

Filter out system alarms from hardware units

The function is used to reduce the number of alarms generated from hardware units, as important alarms tend to disappear in a crowd of alarms.

An example is when commissioning the system or a new part of an existing system, there might be transmitters that are connected and disconnected and the system generates a lot of underflow, overflow and channel error alarms.

The function is configured on the hardware object on the controller. Select **Controller > Hardware AC 800M > Editor > Settings**, then select **Filter out system alarms** from hardware units as shown in the [Figure 118](#).

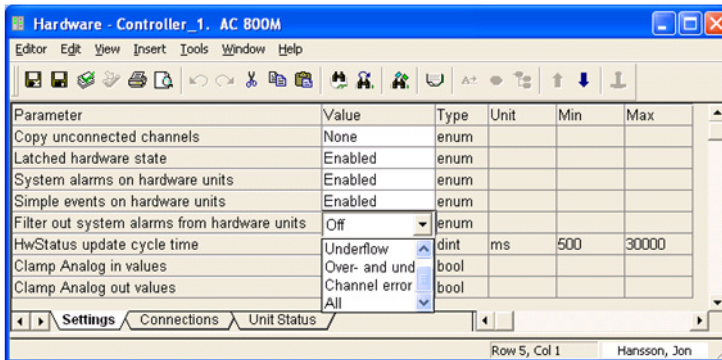


Figure 118. Filter out system alarms

The parameter has five possible values:

Off: The filter function is shut off. The hardware status generates system alarms and systems events for all status changes. This is the default setting.

Underflow: Underflow status changes will not generate any alarms.

Over- and underflow: Neither underflow nor overflow status changes generate alarms.

Channel error: Channel error and IO warning will not generate system alarms or events.

All: Alarms and events from all the status changes above are suppressed.



Even if the setting is set to filter out alarms, the errors and warnings will still be visible in the hardware tree and the Unit Status tab in the hardware online editors. The change to the filter is performed in offline mode and downloaded to the controller to activate the change.



If the function is used during commissioning to decrease the number of alarms, it might be crucial to shut off the filter before entering normal operation.

User Generated System Alarms

User generated system alarms can be defined in your applications via the function block *SystemAlarmCond*.

Handling Alarms and Events

When implementing alarm and event handling, it is very important to create a good system for:

- classifying alarms and events,
- setting the severity of different types of alarms,
- indicating the source of an alarm or event,
- naming alarm conditions.

The most obvious reason for this is that you will be able to create an operator environment in which the operator will quickly be alerted to various things that require attention. The operator will also be able to quickly obtain additional information and decide on the best course of action.

However, alarms and events are also logged, in order to be used for trouble-shooting, and when analyzing things in order to improve performance of the plant.

This subsection describes:

- How to send data in XML format, see [Simple Events](#) on page 286. This is useful when creating batch records.
- How to handle system alarms and events, see [System Alarms and Events](#) on page 286.
- Internal, remote, and external time stamps (Sequence of Events, SOE), including time synchronization, see [Time Stamps](#) on page 286.

Simple Events

The `DataToSimpleEvent` function block is used to send data in XML format, for example, to record data for batch processes.

For more information on how to use this function block, see online help. For examples on how to use the `DataToSimpleEvent` function block, see [Alarm Examples](#) on page 297.

System Alarms and Events

The handling of system alarms and events is to a certain degree configurable. The function block `AttachSystemAlarm` can be used to retrieve information on system alarms and events, such as state, and whether the alarm has been disabled or acknowledged.

The function block `SystemAlarmCond` can be used to retrieve system alarms and events via the application.

Time Stamps

When an alarm or event is created, a time stamp can be added to it, showing the exact time when the event occurred. There are three types of time stamp:

- Internal Time Stamps, that are created by the controller.
- Remote Time Stamps that are read from external communication partners via the parameter `TransitionTime`.
- External Time Stamps that are created by an I/O unit and transferred together with the event.

The `TransitionTime` parameter (of type `date_and_time`) can be used to read a remote time from a remote partner, via other protocols than MMS. The parameter is read each time a change is detected in the monitored signal. If it is left unconnected, it will have no effect.



When adding remote time stamps, it is possible to add any time. However, settings in the operator interface might filter out alarms and events with times that are outside the “normal” range (in the future or far back).

Internal time stamps simply show when the execution cycle in which the alarm was created started. External and remote time stamps show the actual time at which the alarm condition occurred in the external device or partner. All time stamps have a resolution of 1 ms; however, it is the interval time of the task where the alarm function block or module runs that determines the accuracy of the internal time stamps. All alarm function blocks and modules in the same task are given the same time stamp, if activated concurrently.

This is the point of using external and remote time stamps. Internal time stamps can never be more accurate than the execution time of the task allows for. With external or remote time stamps, the accuracy of the time-stamping mechanism in the external or remote device (for example, an S800 I/O unit) sets the limit, something which could seriously improve the accuracy of the time given in entries with external or remote time stamps.

If external time stamps are to be used, the external time stamp parameter (*ExtTimeStamp*) has to be set to True. When using external time stamps, there is also a *SignalId* parameter that is used to indicate the source of the external alarm or event.



External time stamps can only be created by external units with Sequence-of-Event (SOE) support.

All time stamps use UTC (Coordinated Universal Time).

Clock Synchronization

For time stamps to be useful, the whole system must use the same time, that is, the time must be synchronized. See also Clock Synchronization in the *AC 800M Communication Protocols (3BSE035982*)*.

Depending on the type of controller, clock synchronization is possible by four different protocols: CNCP, SNTP, MB 300 TS, and MMS Time Service. Clock synchronization is set up in the controller hardware editor.

It is important to understand the difference between accuracy and resolution when calculating how much a time stamp may deviate from the true system time:

- *Resolution* is the number of decimals that are used to write the time. If the time is given as, for example, 2004-02-19 19:43:22:633, the resolution might be 1 ms (but could also be, for example, 0.5 ms).
- *Accuracy* is a measure of how accurate a time stamp is, that is, how much it may deviate from the true system time. If the accuracy is 1 ms, then 2004-02-19 19:43:22:633 actually means any time between 2004-02-19 19:43:22:632 and 2004-02-19 19:43:22:634.



For a more detailed, conceptual description of time synchronization, see the *AC 800M Communication Protocols (3BSE035982*)* and the *System 800xA Network Configuration (3BSE034463*)*. For information on how to set up time synchronization on a controller, see online help for the processor unit (PM unit) in question.

It is also important to understand that the accuracy deteriorates if a time stamp is created in a unit that is supplied with the time from a controller, via ModuleBus.

The possible difference between the time stamps of two events that occurred at exactly the same time, but in two different units in two different controllers, is the sum of the accuracy of time synchronization in the network and two times the accuracy of the ModuleBus time synchronization.

This means that the difference between external time stamps can be far greater than the accuracy of time synchronization between controllers.

The highest accuracy is achieved by using the CNCP protocol, with an AC 800M controller as master.



Units with SOE require time synchronization throughout the system, see [Clock Synchronization](#) on page 288. The time used by units on ModuleBus is based on the synchronized time received by the controller, but the accuracy is somewhat lower. For information on the accuracy of SOE time stamps, see S800 I/O documentation.

Alarm and Event Communication

Alarm and event information is communicated throughout the control network via OPC servers, that is, a number of OPC Server for AC 800M. When the state of an alarm condition changes, an event notification is sent to all subscribing OPC servers, which then forward these notifications to their clients. Changes in alarms in the OPC server are also forwarded to its clients. Clients can be third party OPC clients, or an 800xA operator station.



For detailed information on how to configure OPC Server for AC 800M, refer to the *AC 800M OPC Server (3BSE035983*)*.

Subscriptions

An OPC server subscribes to event notifications from a control system. Each controller compiles an internal list of all servers interested in various events. Condition-related events are generated when alarm conditions change their state. Simple events can be generated, for example, by the start of a motor. When an event occurs, the control system sends event notifications to all servers on the subscription list.

Configuration of OPC AE Communication – Overview

The whole system for transferring alarms and events, that is, controllers, OPC servers, and OPC clients, must be configured so that there are no disturbances in the alarm and event traffic.

There are several basic rules regarding system configuration:

- A control system can send data or event notifications to one or two subscribing OPC servers.

- A maximum of seven OPC clients can subscribe to data or event notifications from the same OPC server.
- A maximum of four Ethernet links (two redundant) are supported via Ethernet cards.
- A maximum of four Point-to-Point Protocols (PPP) are supported via serial cards.

The OPC server must be configured to recognize the control systems it is to communicate with. The OPC client must be configured to recognize the OPC server(s) it is to communicate with. See [Figure 119](#).

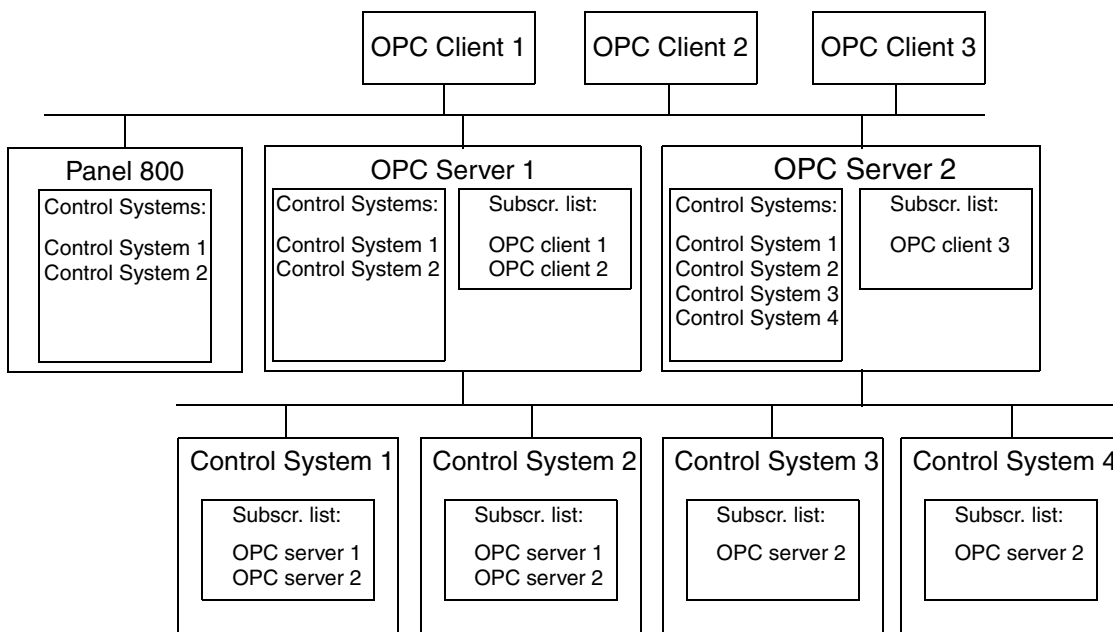


Figure 119. Example of a control network configuration.

Information about how to configure individual OPC servers is found in the *AC 800M OPC Server (3BSE035983*)*, and in the online help, which can be opened from the OPC server panel.

Buffer Queues

For each connected OPC Alarm and Event client, there is an OPC Alarm and Event Server queue. All data passing the OPC Server, such as event notifications, will also pass this queue. [Figure 120](#) shows a control system buffer configuration example, where OPC clients subscribe to alarms and events from different OPC Servers. When a buffer is full, a system simple event is sent upwards to the 800xA System. All buffers are created in accordance with OPC server and CPU settings. Also, see [System Diagnostics](#) on page 316.

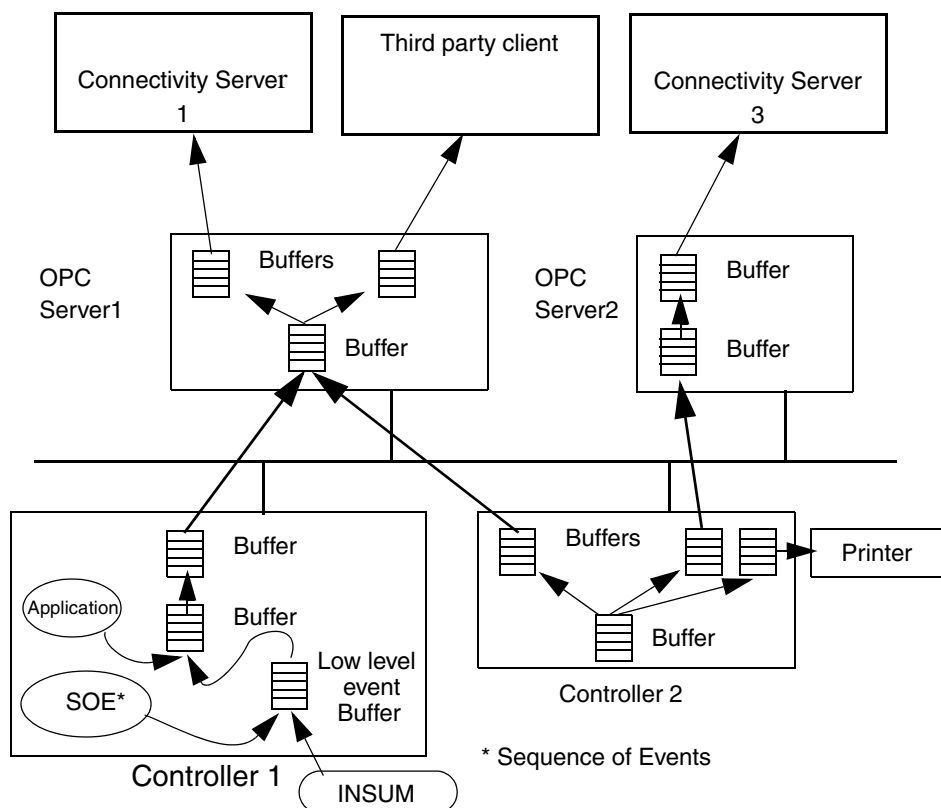


Figure 120. Example of buffer configuration. When a buffer is full, a system simple event is sent upward to the 800xA system and the third party client. In Controller 1, there is also a low level event buffer receiving events from an external device (in the example, an INSUM device).

Buffer Configuration

Alarm and event handling requires a number of buffers. The memory for these buffers must be allocated in the controllers. These settings have to be made in the Settings tab for each controller CPU.

[Table 22](#) describes the parameters in the Settings tab that need to be configured for the buffer. See also [System Diagnostics](#) on page 316.

These settings affect the Available memory. For more information regarding Available memory, refer to the *System 800xA System Guide Technical Data and Configuration (3BSE041434*)* manual.



For controller types with limited memory, the settings for the buffer configuration should be carefully chosen or else the memory becomes full.

Table 22. Memory planning for buffer configuration

Parameter	Comment
AE Local printer event queue size	Each position allocates approximately 300 bytes of memory. The total memory need for local printers is: $300 * \text{AE Local printer event queue size} * \text{AE Max number of local printer event queues}$
AE Max number of local printer event queues	The maximum number of event queues in the controller
AE Event subscription queue size	Each position allocates approximately 300 bytes of memory. Total memory need for subscribing OPC Servers are: $300 * \text{AE Event subscription queue size} * \text{AE Max number of event subscriptions}$
AE Max number of event subscriptions	Number of subscribing OPC Servers

Table 22. Memory planning for buffer configuration (Continued)

Parameter	Comment
AE Buffer size of low level event	Each position allocates 72 bytes of memory. Total memory need for Sequence of Events are: 72 * AE Buffer size of low level event Set this setting to 2 if Sequence of Events is not used
AE Max no of Name Value items	The maximum number of XML tagged events
AE Max percent of log strings	The percentage of Name Value items that are strings. Used to allocate memory for Name Value item strings.

Local Printers

A local printer can be connected to the serial port of a controller, and print out event lists and/or alarm lists as needed.

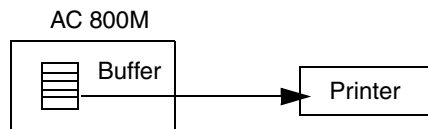


Figure 121. Example of a controller and local printer configuration.

There can be only one local alarm/event printer connected to each controller. Additional printers are invalid. There is limited data flow support for alarm/event printers connected to controllers. Alarms and events that occur when the printer is offline may not be printed when the printer goes online again. This applies to all printers with direct connection to a controller.

Print Format

The print format for alarm conditions and events is governed by a special format syntax.

The system supports the 8-bit ASCII character set (according to Windows). This means that the serial and parallel printers must support the 8-bit character set.

The abbreviations used in these format strings are given in [Table 23](#). The character length of each field is given within parentheses..

Table 23. Abbreviations in format strings

Abbreviation	Explanation of the identification parameters
Ti	Time stamp (MM-DD-HH:MI:SS)
Sr	Source name (maximum 30)
Co	Condition name (maximum 15)
Me	Message (maximum 70)
Cs	Condition state text (maximum 20)
Tt	Transition type text (maximum 20)
S	Severity (4)
C	Class(4)

The fields may be in any order.

Ti, Sr, Co, Me, Cs, and Tt have user-defined dynamic lengths. If the length of a string is defined as longer than a presentation function that is already set, the presentation is reduced accordingly

The text for the condition state originates from project constants such as `cAlarmCondStatetext.On1`, `cAlarmCondStatetext.Off1`, and so on.

A maximum of 132 characters can be printed for each alarm/event.

Globally Defined Print Formats

Global print formats are defined in the project constants, which are categorized based on alarm and event conditions or transitions:

- For Alarm Conditions
 - `cPrintAlarmPres.AlarmCondFormat`
 - `cPrintAlarmPres.TimeFormat`

- cPrintAlarmPres.FooterFormat
- For Events
 - cPrintEventPres.CondEventFormat
 - cPrintEventPres.SmpEventFormat
 - cPrintEventPres.TimeFormat
- For Alarm Condition State Texts
 - cAlarmCondStateText.Undefined
 - cAlarmCondStateText.On1
 - cAlarmCondStateText.Off1
 - cAlarmCondStateText.Acked
 - cAlarmCondStateText.Disabled
 - cAlarmCondStateText.Idle
 - cAlarmCondStateText.Autodisabled
- For Event Transition Texts
 - cEventTransitionText.Undefined
 - cEventTransitionText.On1
 - cEventTransitionText.Off1
 - cEventTransitionText.Ack
 - cEventTransitionText.Disable
 - cEventTransitionText.Enable
 - cEventTransitionText.Autodisable

Sending an Alarm to the Application

Instead of sending your alarms to a local printer you can choose to only redirect the alarm to the application. The function block PrintEvents contain two parameters; the first parameter EventItem catch the values (Source Name, Condition name, Time stamp, Severity etc.) and the second parameter EventItemText format these values

as if they was send to a printer and bring it to the application as well. Hence, these values can then be sent and processed by your local code.

However, sending an alarm only to the application requires that you do not connect the Channel parameter (leaving the Parameter field empty).



By sending an alarm to the application you can then redirect this information to your cell phone. Every time an incoming alarm has a severity higher than 700, you should be notified with a SMS.

Third Party OPC Clients

Normally, all OPC traffic is kept within the 800xA system, which integrates all alarm and event function into a single system. However, it is still possible to connect third party OPC clients to the OPC servers, since OPC Server for AC 800M supports the OPC standard. In this case, it is important to know that there are extensions and limitations in relation to the standard. For further information, refer to the *AC 800M OPC Server (3BSE035983*)*.

Translation – NLS Handling of Strings

Translation of alarm and event strings requires that the strings to be translated contain control characters indicating that they should be translated, and that they follow the National Language Support (NLS) syntax.

The operator environment supports NLS handling and this is set up for the operator workplace. When the operator environment discovers a string that uses NLS syntax, it will automatically translate this string to the language that has been set, provided that there is a corresponding string in that language and that the Alarm and Event Translator aspects has been set up correctly.

Translation supports the UNICODE standard and is triggered by two pipe characters (||). Parameters can be used inside the string. The position of each parameter is indicated inside brackets, {1} {2} etc. Parameter values are given at the end of the string, separated by backslashes (\).

A string prepared for translation might look as in the below example:

```
T220      ||PR1_ACOF_sup_time_changed_to_{1}ms_for  
motor_{2}\5\M101\
```


This would result in the following string if no translation is applied:
T220 ACOF supervision time changed to 5ms for motor M101

For more information on NLS syntax, refer to online help for alarm and event handling.

The following strings have NLS support:

- The condition name (parameter *CondName*). Condition names cannot be longer than 15 characters.
- Alarm/event messages (parameter *Message*).



It is also possible to translate the source name (parameter *SrcName*). However, due to the fact that this is normally handled by other functions in the operator interface, this is not recommended.

When printing alarms and events, all NLS control characters are removed.



NLS handling must be setup in the operator workplace. For instructions on how to enter translations and select language, refer to system and operator workplace documentation.

Alarm Examples

The following subsection contains a number of examples designed to help you understand how alarm and event handling works and how to use the types in the Alarm and Event library.

- AlarmSimple_M example shows how function blocks (*AlarmCond*, *SimpleEventDetector*), and control modules (*AlarmCondM*) from the Alarm and Event library can be used, and how different parameters affect the condition state. See [AlarmSimple_M Example](#) on page 298.
- The AlarmSimple_M example can also be used to study the aspects that are generated by alarm conditions. See [Alarm and Event Aspect Example \(AlarmSimple_M\)](#) on page 303.
- The alarm owner concept is illustrated by a couple of examples. See [Alarm Owner Examples](#) on page 304.
- How to set up functions for inhibiting and disabling alarms is shown in [Inhibit Example](#) on page 310.

- There are three examples of how to use simple event data. See [Simple Event Examples](#) on page 312.

AlarmSimple_M Example

The example project AlarmSimple_M is located in the Example folder (under Program Files in Windows) and is installed with the system. Run AlarmSimple_M simultaneously when studying this section.



The example file has the suffix .afw. Browse to the *Example* folder inside the Import/Export function in Plant Explorer, and import the example project to the Control Structure. See the *System 800xA Control AC 800M Getting Started (3BSE041880*)*.

The example contains:

- A motor, named M101, with two supervised out signals: SwitchGearError and M101OverLoad.
 - SwitchGearError has severity 50 and belongs to class 15. This signal is connected to an AlarmCond function block named SwitchGearAlarm.
 - M101OverLoad has severity 100 and belongs to class 50. This signal is connected to an SimpleEventDetector function block named OverLoadEvent.
- Two tanks, named Tank11 and Tank12, both with supervision of the tank level. Each tank contains two alarm conditions, High and Low, which are based on the AlarmCondM control module type.

Figure 122 shows a partial view of the two tanks, Tank11 and Tank12. Figure 123 shows the Project Explorer view of the defined types and control modules.

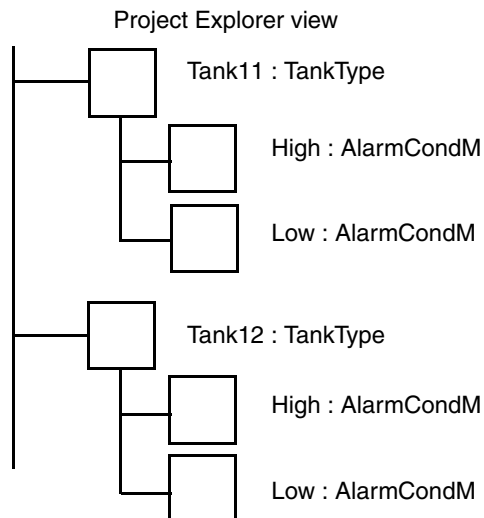


Figure 122. Project Explorer view

This example uses control modules, but function blocks might as well be used.

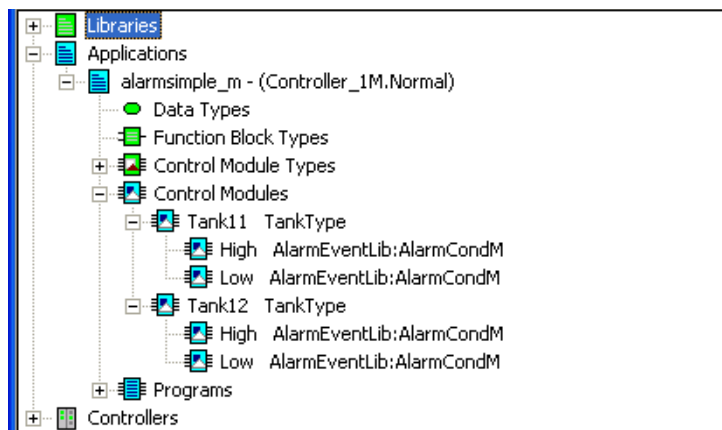


Figure 123. Tank11 and Tank12 in Project Explorer.

Once you have imported the example, you can set it in test mode and study how different parameters affect the behavior of an alarm condition and how simple events are generated:

1. Open the AlarmSimple_M example project.
2. Select **Tools > Test Mode** to enter test mode.
3. Select **View > Expand All** to expand the project tree. The following window will be displayed.
4. Under Applications, alarmsimple_m and Programs, double-click Program2. The online editor is displayed. The two function blocks, one for each supervised signal, are shown under the Function Blocks tab in [Figure 124](#).

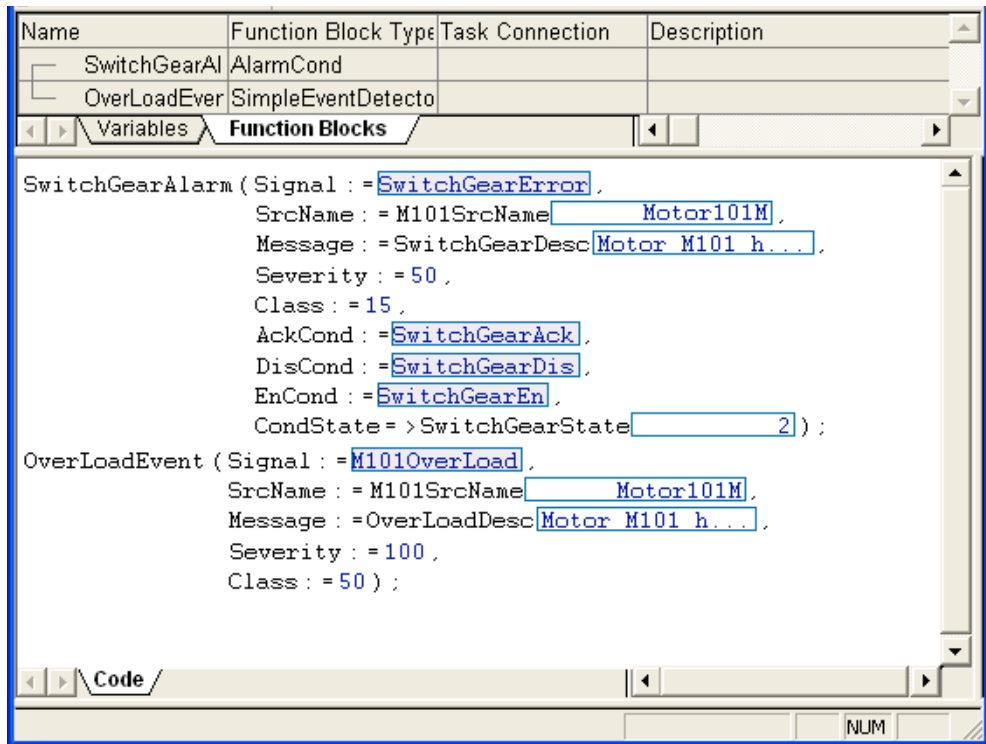


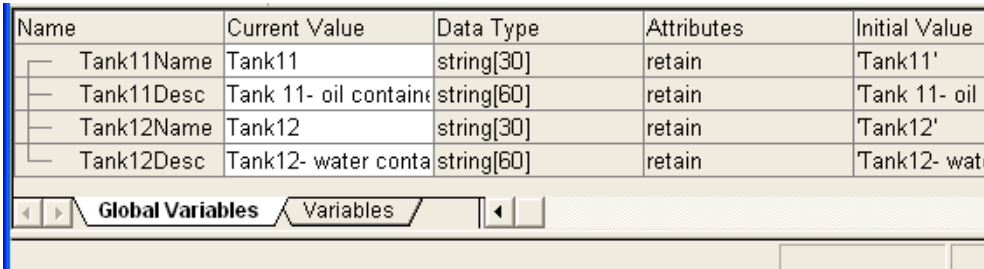
Figure 124. Part of Online editor for Program2.

5. Try changing the below variables (see [Table 24](#)) under the Variables tab or in the code pane, to generate and acknowledge the alarm, or to generate the simple event. Note that in this example, not all parameters are used. Function blocks can be viewed in their online editor. The variable *SwitchGearState* shows the current alarm state.

Table 24. Variables used to generate alarm and events.

SwitchGearError	Supervised signal. Set/reset alarm condition here.
M101OverLoad	Supervised signal. Set/reset simple event here.
SwitchGearAck	Acknowledge alarm here.
SwitchGearDis	Disable alarm here.
SwitchGearEn	Enable alarm here.

6. Double-click alarmsimple_m under Applications to display the corresponding online editor (Figure 125). Here, you can study variables connected to control module parameters.



Name	Current Value	Data Type	Attributes	Initial Value
Tank11Name	Tank11	string[30]	retain	'Tank11'
Tank11Desc	Tank 11- oil containe	string[60]	retain	'Tank 11- oil
Tank12Name	Tank12	string[30]	retain	'Tank12'
Tank12Desc	Tank12- water conta	string[60]	retain	'Tank12- wat

Global Variables Variables

Figure 125. Part of Online editor for alarmsimple_m.

- Under Applications, alarmsimple_m, and Control Modules, double-click Tank11 or Tank12. The corresponding online editor is displayed (Figure 126).

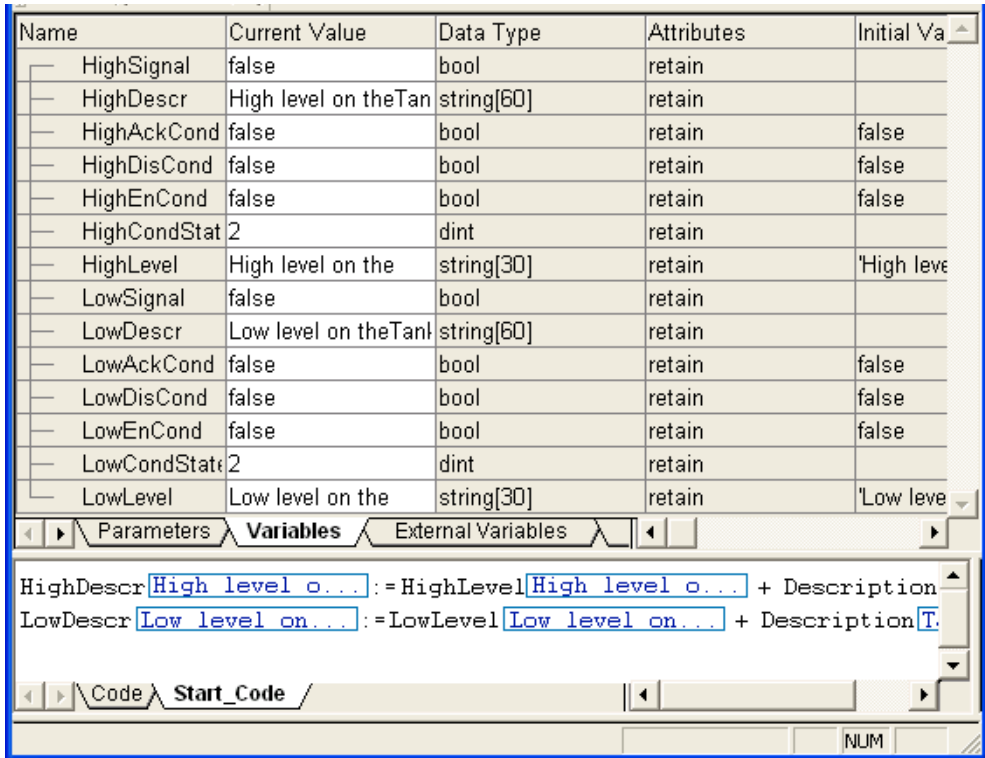


Figure 126. Part of Online editor for Tank11.

- Try to generate, acknowledge, or disable/enable alarm conditions by changing the values of the parameters. Note how the condition state changes (*HighCondState* and *LowCondState*).

When ready, exit test mode and close all windows.

Alarm and Event Aspect Example (AlarmSimple_M)

The Tank objects in the AlarmSimple_M example project are aspect objects. The alarm conditions are collected in one aspect of the tank object. The reason for this is that the control module type *AlarmCondM* has a predefined attribute Alarm. All objects based on types with the Alarm attribute set are shown in the Control Alarm Event aspect of the parent object. The condition name and instance name of an alarm condition module are identical, unless the CondName parameter has been used to set another condition name.

Some Tank11 aspects are shown in [Figure 127](#), which shows part of the Control Structure in Plant Explorer. The Control Alarm Event aspect is created whenever the object has the Alarm Owner attribute set to True. The AlarmList aspect has been added. This aspect has no connection with Control Builder.

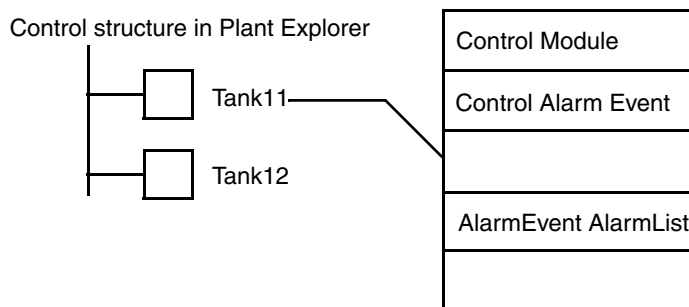


Figure 127. Aspect Object Tank11 in Plant Explorer, showing some aspects.

Condition/Instance names presented in the Control Alarm Event aspect are shown in [Figure 128](#).

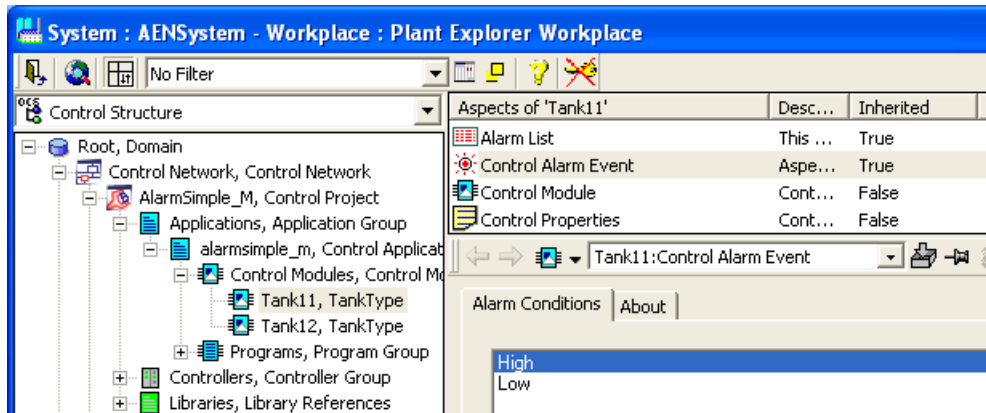


Figure 128. Example project in the Plant Explorer, with alarm conditions shown in the Control Alarm Event aspect.

Alarm Owner Examples

The following examples show how the alarm owner concept can be used to control which object is considered to be the source of an alarm or event.

Figure 129 shows a library called PipeLib:

- PipeLib contains two types, MyMotor_type and MyPipe_type.
- MyMotor_type contains an alarm control module (of the type AlarmCondM).
- MyPipe_type contains two motors of the type MyMotor_type, and two alarm condition control modules (of the type AlarmCondM).

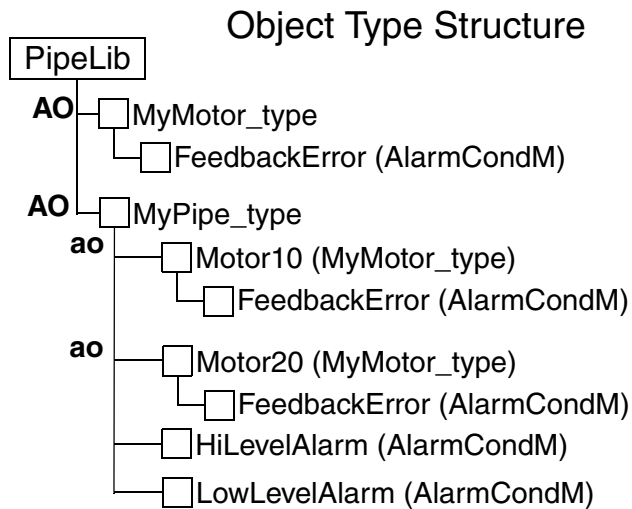


Figure 129. PipeLib. AO=Alarm Owner (setting in type), ao=alarm owner (setting is inherited from type).

We use the PipeLib library and two single control modules (SM1 and SM2) to create a structure containing three tanks of the type MyPipe_type, see [Figure 130](#). We set the Alarm Owner attribute to False for SM1, but to True for SM2.

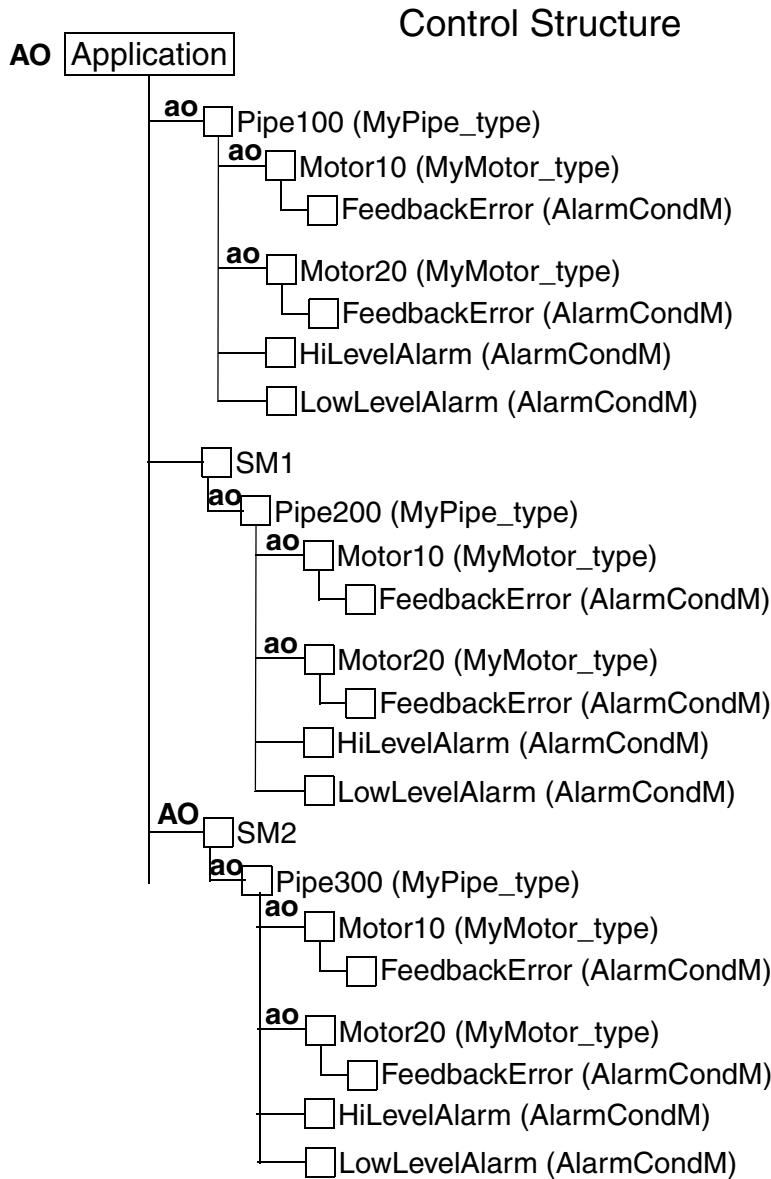


Figure 130. Three pipes with different alarm owner conditions.

What happens if an alarm is created inside this structure? Which object will be the alarm owner? The answer is that the alarm ownership will depend on the existence of an unbroken chain of alarm owners:

- For Pipe100 and Pipe300, the HiLevel and LowLevel alarms will be associated with the pipe, since there is an unbroken chain of alarm owners from the tanks, up to the application.
- For Pipe200, the HiLevel and LowLevel alarms will be associated with the application, since there is no unbroken chain of alarm owners leading from the application down to the pipe.
- For Pipe100 and Pipe300, FeedbackError alarms from the motors will be associated with the motor in question, since there is an unbroken chain of alarm owners from each motor, up to the application.
- For Pipe200, FeedbackError alarms from the motors will be associated with the application, since there is no unbroken chain of alarm owners leading from the application down to the motors.



It is easy to manipulate the alarm ownership. The alarm owner chain can always be broken by inserting a “blind object” which is not an alarm owner. For example, inserting such an object between Pipe100 and Motor10 in the above example would cause FeedbackError from Pipe100 to be the owner of FeedbackError alarms from Motor10, while Motor20 would still be the owner of FeedbackError alarms from Motor20. See [Figure 131](#).



The situation where all alarms from the SM1 single module have the application as alarm owner is of course not desirable. It is simply included to illustrate what happens when the alarm owner chain is broken.

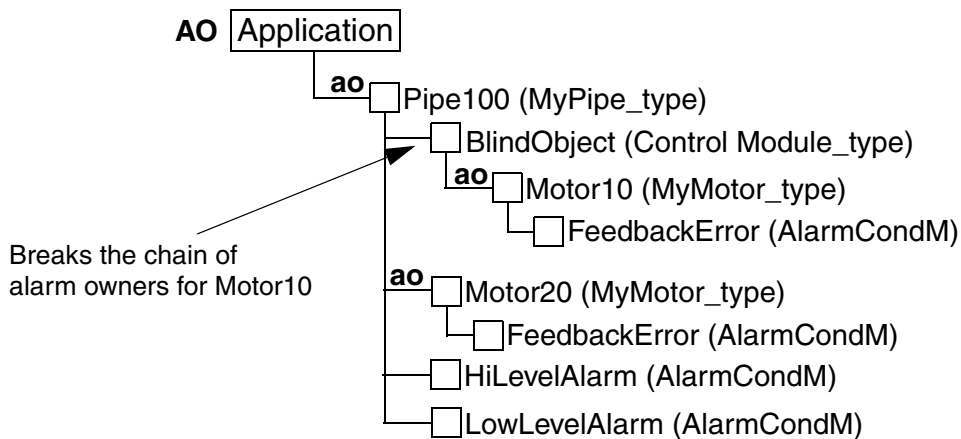


Figure 131. Inserting a “blind object” to break the alarm owner chain.

Condition State Example

The following example shows how to use the condition state parameter (*CondState*) to control a pump.

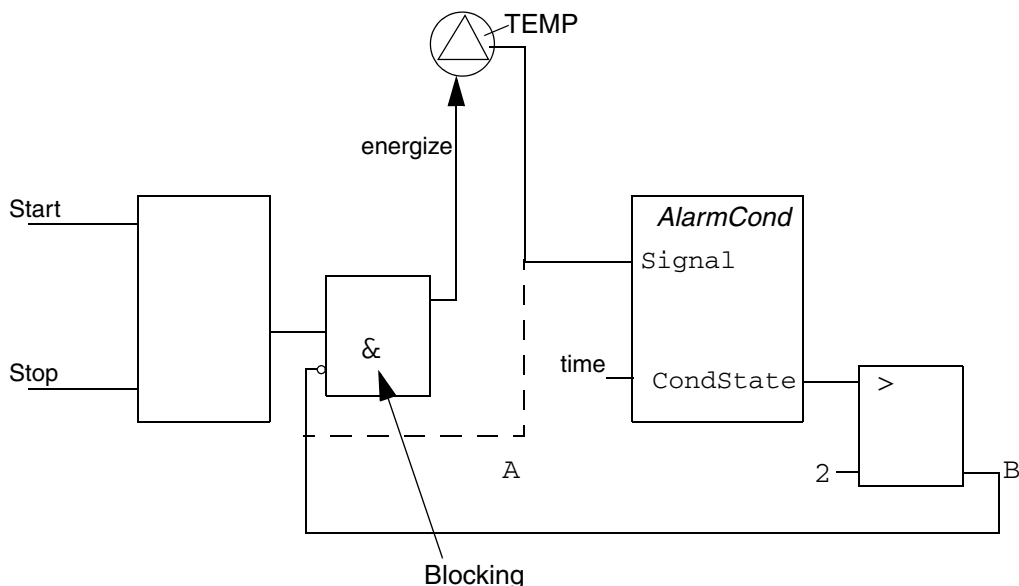


Figure 132. Manipulating the condition state using I/O.

Figure 132 shows two alternative ways of stopping a pump when the temperature is too high. The TEMP signal goes high when the temperature is too high.

In alternative A, the TEMP signal is simply used to stop the pump (using the blocking function, note that the TEMP input is inverted). There is no way to disable this alarm. The pump is blocked as long as TEMP is high.

Alternative B uses an AlarmCond function block, which makes it possible to wait for an action from the operator, before unblocking the pump. The blocking signal to the pump does not go high until *CondState* > 2, that is, the alarm is enabled and not idle (for a list of possible states, see [Status Information](#) on page 266). Once it has gone high, it does not go low until *Condstate* => 1, that is, the alarm is disabled or has returned to its idle state (this means that the alarm must be acknowledged by the operator and TEMP must go low before the pump is unblocked, as long as acknowledgement rule 1 is used).

Alternative B also makes it possible to disable the blocking function by simply disabling the alarm condition.



This example has been simplified to illustrate a principle. In reality, it would not be desirable to have a motor start when an alarm is acknowledged. Instead, the operator would acknowledge the alarm, and then start the motor with a separate command.

Inhibit Example

The below example shows how to implement the inhibit function for a motor M103 (see [Figure 133](#)):

- An oil pressure sensor, P103, is used to stop the motor M103 if the oil pressure is too low.
- A `SignalInReal` object is used to supervise the sensor and a `MotorUni` is used to control the motor.
- The `LTLAct` output from `SignalInReal` is connected to the `PriorityCmd01` in `MotorUni`. This means that the motor will be forced to stop when the oil pressure is below the LL level. `LTLStat` may be connected to a warning lamp in a panel.

During start up of the equipment it is known that the oil pressure will be below the limit, but it must be possible to start the motor. Therefore, the application logic will set the `EnableLL` parameter in `SignalInReal` to `False` during start-up. This means that `LTLAct` will not be set, that is, the motor will not be stopped and no alarm is sent to the alarm list as long as the motor is starting up. `LTLStat` will not be set and the lamp will not be lit.

Suppose the operator, maybe for testing, wants to run the equipment at an oil pressure below the LL level. He could then inhibit `SignalInReal` from the faceplate. The motor will still run during the test, but an alarm will be sent to the alarm list. `LTLStat` will be set and the lamp will be lit.

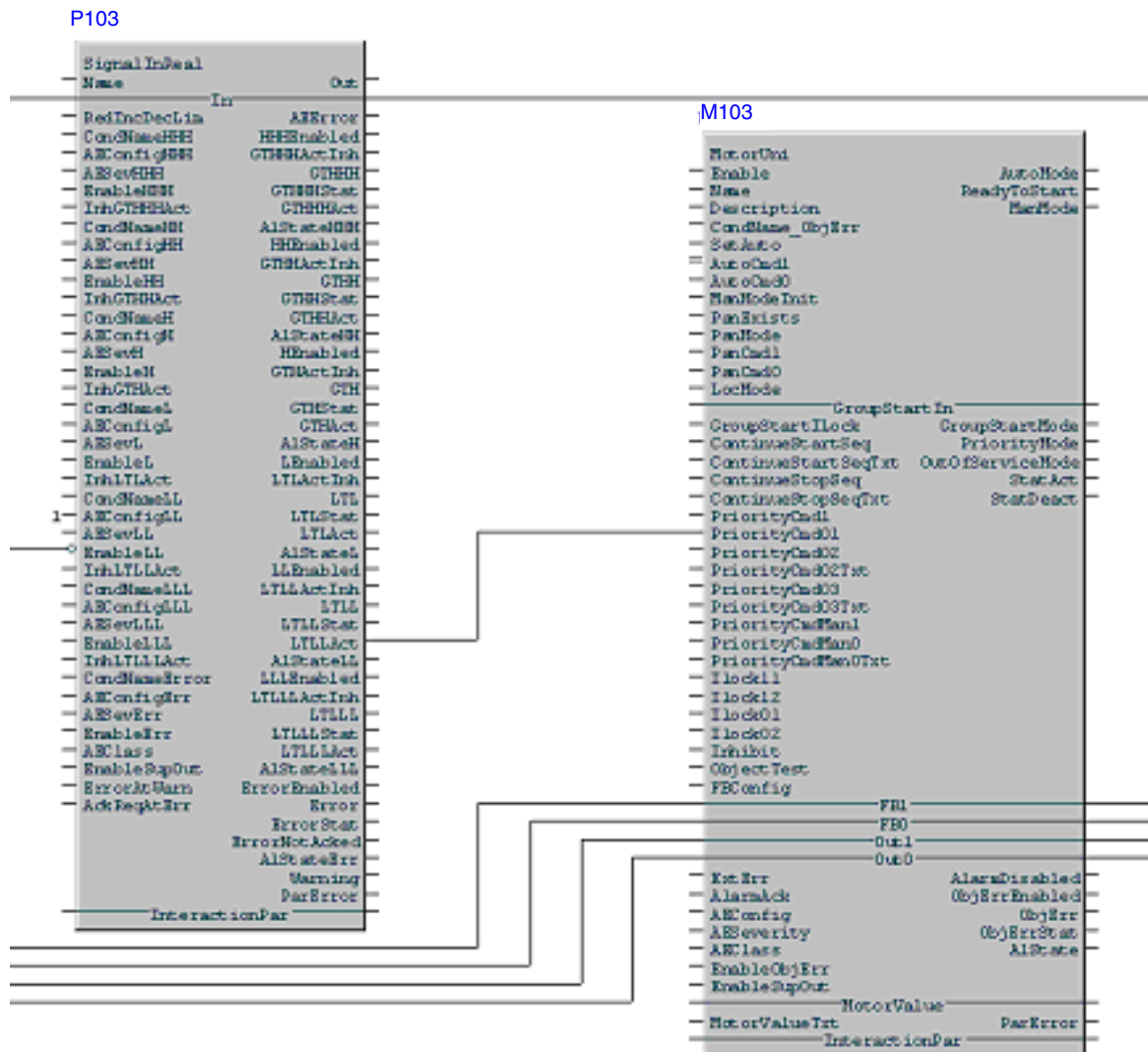


Figure 133. Example of how to implement inhibition of an alarm.

Simple Event Examples

The below examples show how to use the DataToSimpleEvent function block to send simple event data, for example for a batch process, where data records should be generated for the process at a number of points. There are three examples:

- [Simple Data](#) on page 312,
- [Structured Data – Example 1](#) on page 314,
- [Structured Data – Example 2](#) on page 314.

Simple Data

Presume that an engineer wants to record three parameters in the process: a temperature, a pressure and a stirring rate. Consequently, the engineer names them:

```
varTEMP = "TEMP"
```

```
varPRESS = "PRESS"
```

```
varSTRAT = "STRAT"
```

These are the names the user wants to see on the screen when the recording is done, but these names are not the same as the variable names. Instead, the names are coupled to the extensible parameters in the *Name* field:

```
Name[1] = varTEMP
```

```
Name[2] = varPRESS
```

```
Name[3] = varSTRAT
```

During execution TEMP=300.2, PRESS=23.1, and STRAT=10. Temp and press are real values (real) and STRAT is an integer, which causes no problem since *Values* is of AnyType.

NestingLevel "1" is chosen and this is how it could look in Control Builder:

```
varTEMP = "TEMP"  
tempValue := 300.2;  
pressValue := 23.1;  
My Log(SrcName := SrcName,  
      Message := Message,  
      Class := Class,  
      EventCode := thisNbrEvent  
      RecipePath := myLongPath,  
      Status => Status,  
      Name[1] := varTEMP,  
      Value[1] := tempValue,  
      NestingLevel[1] := 1,  
      Name[2] := varPRESS,  
      Value[2] := pressValue,  
      NestingLevel[2] := 1,  
      Name[3] := varSTRAT,  
      Value[3] := stratValue,  
      NestingLevel[3] := 1 );
```

In OPC Server for AC 800M, this will be encoded into an XML string.

```
<DATA_EV_LOG>  
  <TEMP Value="300.2" type="real"/>  
  <PRESS Value="23.1" type="real"/>  
  <STRAT Value="10" type="int"/>  
</DATA_EV_LOG>
```

Structured Data – Example 1

An engineer wants to record data that belong together, that is, he or she wants to create a structure named PHYS_DATA containing physical properties of an object, in this case a tank.

The structure (PHYS_DATA) has no value in itself and the *NestingLevel=1* when PHYS_DATA is coupled to the first extensible parameter.

The next step is to give PHYS_DATA properties, and three components are created in the following three extensible parameters:

height=4.1

length=3.0

depth=1.0

Since the parameters above are physical properties of PHYS_DATA, they are assigned with *NestingLevel=2*. They are all floats.

In this case, the XML data in OPC Server for AC 800M will look like:

```
<DATA_EV_LOG>
  <PHYS_DATA Value="" type="">
    <height Value="4.1" type="real"/>
    <depth Value="3.0" type="real"/>
    <length Value="1.0" type="real"/>
  </PHYS_DATA>
</DATA_EV_LOG>
```

Structured Data – Example 2

In this example, the engineer is in the same situation as in the previous example, but now he or she also wants to record the recipe parameters in one of the batch objects. The same procedure as in **Example 1** is performed but a new parameter “RecipePar” is added and *NestingLevel=-1* is set. With *NestingLevel=-1* it is indicated that the recipe parameters to be fetched are placed on *NestingLevel=1*, since the height, depth, and length values in the previous example were to be placed on *NestingLevel=2*.

The recipe parameters are fetched in the controller and are:

heat=3.4

temp=349.4

heating=true

From a *Control Builder M* view, this would look like:

```
structName := "PHYS_DATA";
varHeight := "height";
heightValue := 4.1;

varRecipe := "RecipePar"

LogThis(SrcName := SrcName,
        Message := Message,
        Severity := Severity,
        Class := Class,
        EventCode := thisNbrEvent,
        RecipePath := myLongPath,
        Status => Status,
        Name[1] := structName,
        Value[1] := EmptyValue,
        NestingLevel[1] := 1,
        Name[2] := varHeight,
        Value[2] := heightValue,
        NestingLevel[2] := 2,
        Name[3] := varDepth,
        Value[3] := depthValue,
        NestingLevel[3] := 2,
        Name[4] := varLength,
        Value[4] := lengthValue,
        NestingLevel[4] := 2,
        Name[5] := varRecipe,
        Value[5] := EmptyValue,
        NestingLevel[5] := -1 );
```

The XML data will look as below. The last three parameters are fetched from a Batch Object.

```
<DATA_EV_LOG>
  <PHYS_DATA Value="" type="">
    <height Value="4.1" type="real"/>
    <depth Value="3.0" type="real"/>
    <length Value="1.0" type="real"/>
  </PHYS_DATA>
  <RecipePar Value="" type=""/>
  <heat Value="3.4" type="real"/>
  <temp Value="349.4" type="real"/>
  <heating Value="true" type="bool"/>
</DATA_EV_LOG>
```

Alarm and Event Functions

There are a number of functions that can be used to analyze and supervise alarm and event handling:

- The function block `SystemDiagnostics` contains a part that displays alarm and event related information. See [System Diagnostics](#) on page 316.
- For those who need detailed information about the alarm and event state machine, there is a collection of state diagrams. See [Acknowledgement Rules – State Diagrams](#) on page 317.

System Diagnostics

When in online mode, it is possible to view information regarding memory via the interaction window of the function block `SystemDiagnostics` (located in the Basic library).

The advanced mode of the interaction window displays system memory information.

There is also an **Alarm and Event** button which, if clicked, displays information regarding:

- Used amount of buffer size,
- The number of:
 - a. alarms in the controller,
 - b. different condition names in the controller,
 - c. local printer queues,
 - d. subscribing OPC Servers.
- The IP-addresses of the subscribing OPC Servers.

Acknowledgement Rules – State Diagrams

The control system handles four different condition state diagrams according to five different acknowledgement rules.

Acknowledgement Rule 1

Rule number 1 uses three different state diagrams.

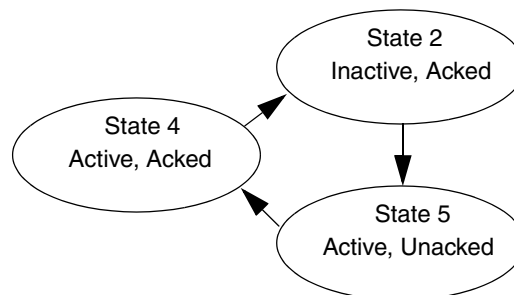


Figure 134. State diagram for enabled alarm conditions with AckRule 1, part 1.

In [Figure 134](#) above, the alarm is in its normal state when it becomes active. It is then acknowledged, and on becoming inactive it returns to its normal state.

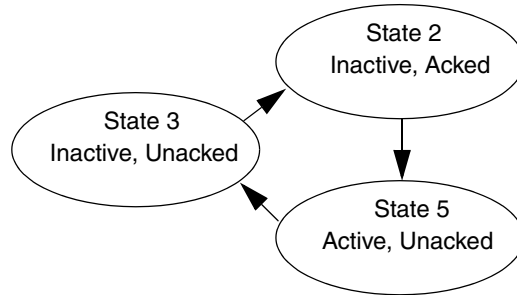


Figure 135. State diagram for enabled alarm conditions with AckRule 1, part 2.

In Figure 135 above, the alarm is in its normal state when the alarm becomes active. It then becomes inactive, and on being acknowledged returns to its normal state.

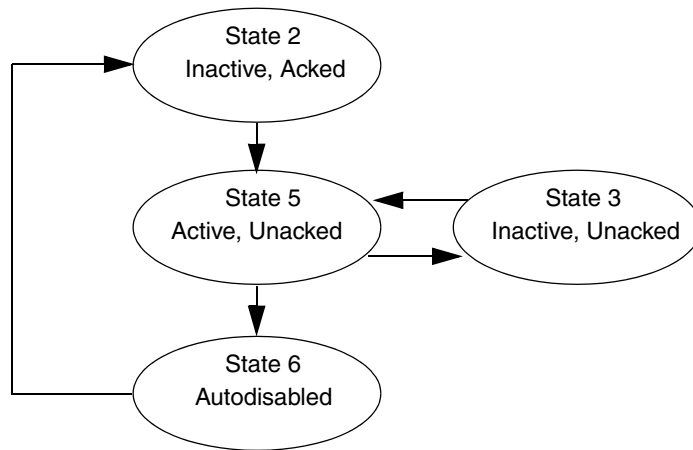


Figure 136. State diagram for enabled alarm conditions with AckRule 1, part 3.

The third instance occurs when an alarm switches between active and inactive without being acknowledged. In Figure 136, the alarm starts in its normal state and becomes active. It then switches twice between active and inactive without being acknowledged. When the alarm becomes inactive a third time it is automatically placed in the Auto-disabled state. Whether the alarm is active or inactive in this state is of no significance. When acknowledged the alarm returns to its normal state.



The default setting for auto-disable is three times. This can be changed through the CPU setting *AE Limit Auto Disable*. If it is set to 0, there will be no auto-disable function. There is also a system variable called *AlarmAutoDisableLimit* which affects all process alarms with acknowledgement rule number 1 (*AckRule=1*).

Acknowledgement Rule 2

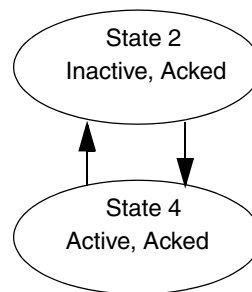


Figure 137. State diagram for enabled alarm conditions with AckRule 2.

Alarm conditions with AckRule 2 does not require acknowledgement and therefore follow a different state diagram. When the alarm becomes active it switches to an active and acknowledged state. On becoming inactive it returns to its normal state.

Acknowledgement Rule 3

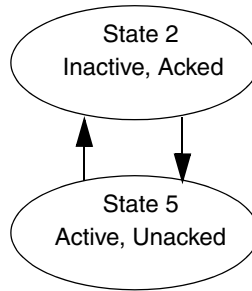


Figure 138. State diagram for enabled alarm conditions with AckRule 3.

Regardless of the signal being monitored, alarm conditions with AckRule 3 changes immediately to its normal state on acknowledgement. The alarm is no longer active and disappears from the alarm list provided by an OPC client.

Acknowledgement Rule 4

Presently, Acknowledgement Rule 4 (AckRule 4) is reserved for future use.

Acknowledgement Rule 5

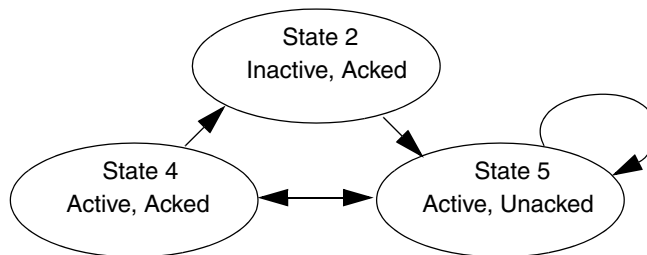


Figure 139. State diagram for enabled alarm conditions with AckRule 5, part 1.

AckRule 5 is used for so called sum system alarms. System alarms associated with hardware units are typical examples of sum system alarms. They are used to indicate several different errors that occur at the same time.

There are two procedures for sum system alarms, that is, for AckRule 5. The first of these is described in Figure 139 above. The sum system alarm is in its normal state when it becomes active. Sum system alarms are used as a collection of errors and Acknowledgement means that all errors are acknowledged. On becoming inactive it returns to its normal state.

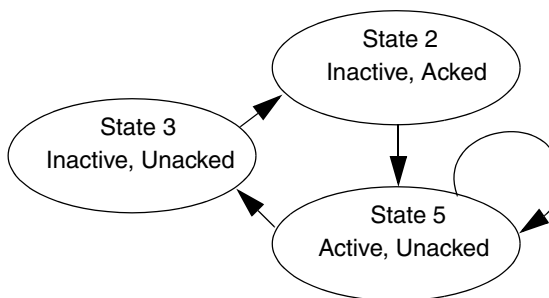


Figure 140. State diagram for enabled alarm conditions with AckRule 5, part 2.

The second instance is shown in Figure 140 above. The sum system alarm is in its normal state when it becomes active. It then becomes inactive, and on being acknowledged returns to its normal state.

Any alarm can be disabled from any state, and when re-enabled placed in the Inactive and Acked state. If the alarm state engine receives an incorrect Enable, Disable or Acknowledgement request, the request is ignored.

Alarm Shelving

The alarm shelving function allows operators to temporarily handle undesired alarms. Shelving occurs for a specified time limit, and calls for operator attention when that time has elapsed.

Alarm Shelving allows the operator to customize alarm settings temporarily in specific situations, and avoids unintended change to the alarm system design. This also retains operator effectiveness and helps improve overall plant reliability.

Hidden alarms and disabled alarms cannot be shelved.

For more details about Alarm Shelving refer to *System 800xA Operation (3BSE036904*)* and *System 800xA Configuration (3BDS011222*)* manuals.

Section 3 Communication

Introduction

This section describes how to configure communication throughout your control network. How to design your control network, and which protocol(s) to choose for this is described in the *AC 800M Communication Protocols (3BSE035982*)*.



Special restrictions apply to communication with SIL certified applications, see the *System 800xA Safety AC 800M High Integrity Safety Manual (3BNP004865*)*.

This section is split into the following parts:

- [Communication Libraries](#) on page 324 gives a brief overview of the Communication standard libraries.
- [Supported Protocols](#) on page 340 gives a brief overview of the protocols supported by control builder.
- [Control Network](#) on page 341 describes Control Network, which is used to communicate between controllers, engineering stations, and external devices.
- [Variable Communication](#) on page 343 describes variable communication briefly, and contains references to more detailed information.
- [Reading/Sending Data](#) on page 348 describes reading and sending data.
- [Fieldbus Communication](#) on page 355 describes the supported fieldbus protocols briefly.
- [HART Communication](#) on page 358 describes HART support (tool routing).
- [SIL Certified Communication](#) on page 358 describes communication between SIL certified applications, both between High Integrity controllers and between applications residing in the same High Integrity controller.

Communication Libraries

The Communication libraries contains a number of libraries, one for each protocol, with function block types for reading and writing variables from one system to another. Typical communication function block types are named using the protocol name and function, for example, COMLIRead or INSUMConnect.



All supported protocols are described in the *AC 800M Communication Protocols (3BSE035982*)*, which also contains general information about how to set up communication in a control network. For detailed information on how to connect and configure function block types and control module types, see the corresponding online help (select the type and press F1).

COMLI Communication Library

The COMLI Communication library (COMLICommLib) contains function block types and data types for COMLI communication.

COMLI function block types follow the IEC 1131 standard, but some divergences occur. COMLI can be used for point-to-point or multidrop communication. Communication takes place serially and asynchronously, based on the master/slave principle, and in half duplex. Only address-oriented COMLI is supported on serial channels.

Foundation FIELDBUS HSE Communication Library

The Foundation FIELDBUS HSE Communication library (FFHSECommLib) contains data types, function block types and control module types for FOUNDATION Fieldbus HSE (FF HSE) communication. Types from the library can be used for direct communication with FF HSE devices via CI860, or to create a FF HSE Link system, using CI860 communication units to communicate with the controller.

Types from the FF HSE communication library can be used for:

- Publisher/Subscriber (also called Subscriber/Provider) communication, see [Publisher/Subscriber Communication](#) on page 353 (for FF HSE, this method only allow communication using the data types DS65 and DS66).

Function Block Types

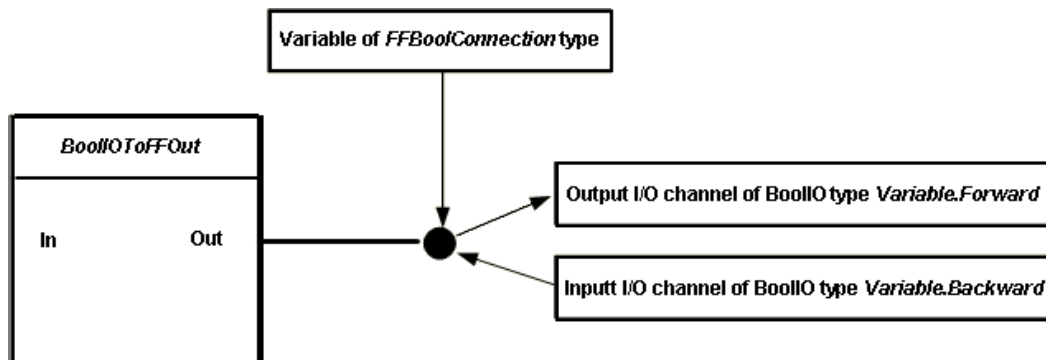


Figure 141. BoolIOToFFOut

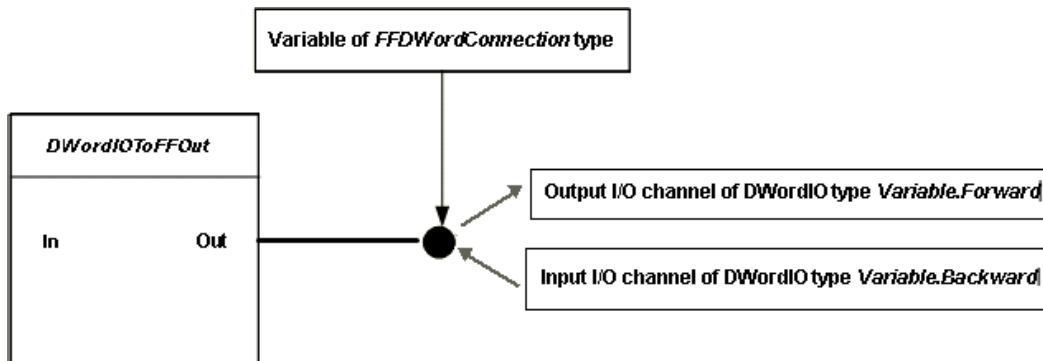


Figure 142. DWordIOToFFOut.

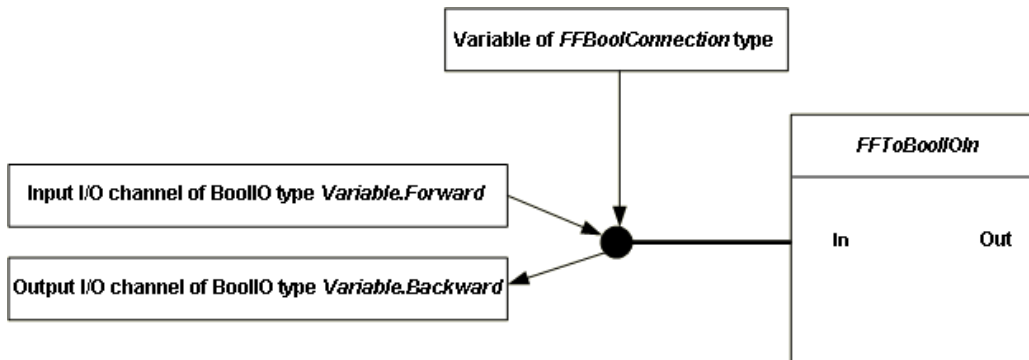


Figure 143. FFToBoolIOIn

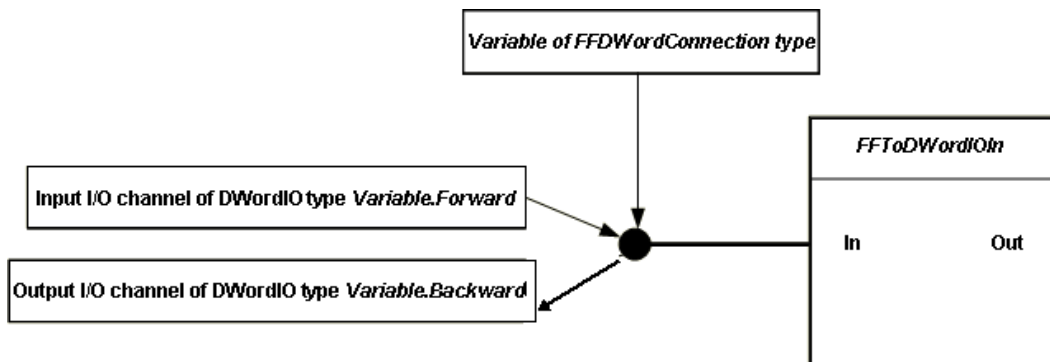


Figure 144. FFDWordIOIn

Control Module Types

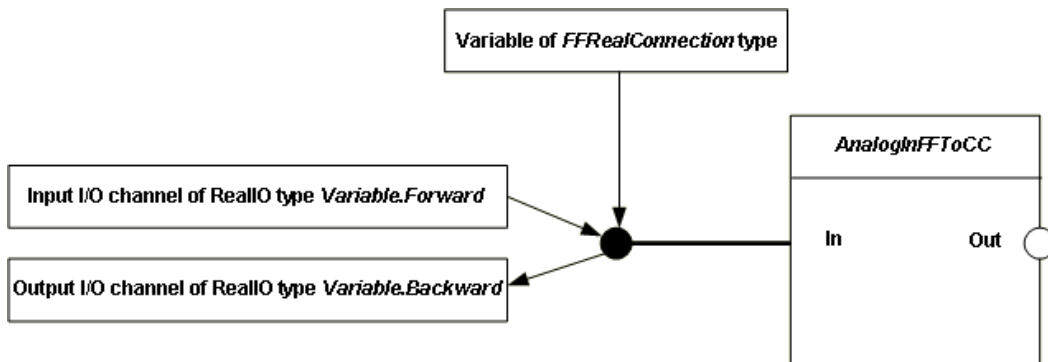


Figure 145. AnalogInFFToCC

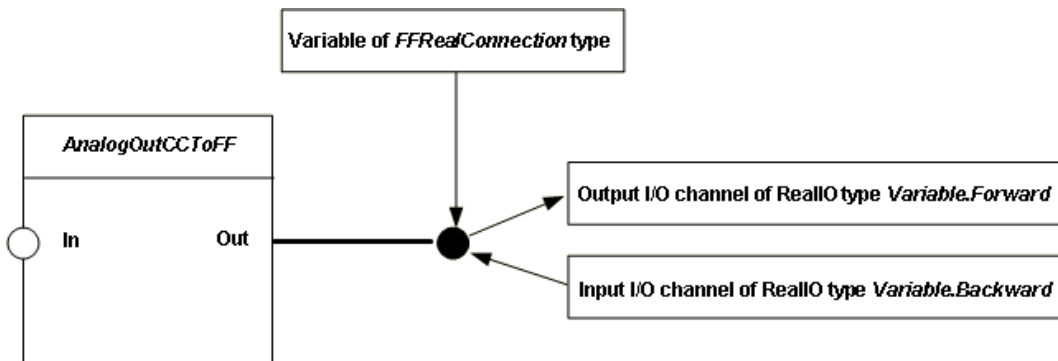
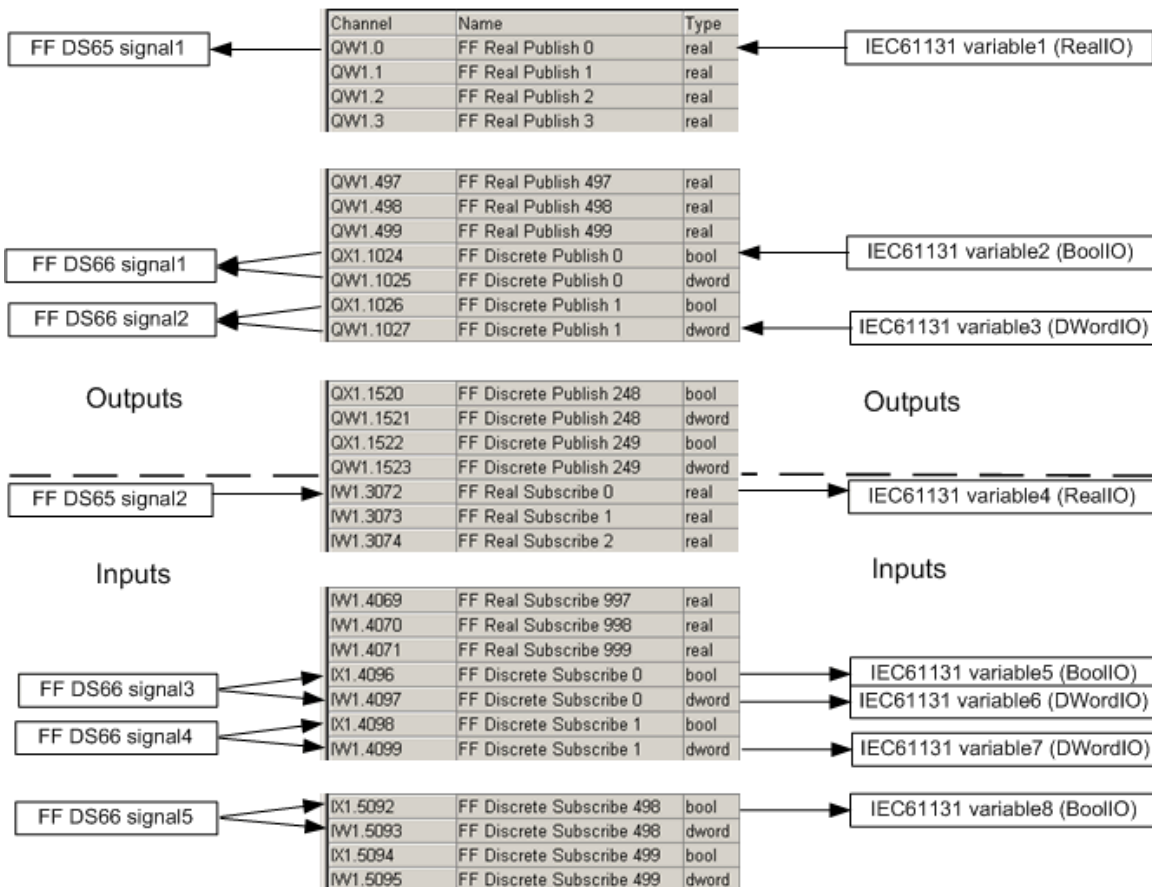


Figure 146. AnalogOutCCToFF

Examples of FOUNDATION Fieldbus HSE Connections

The below figures show examples of FOUNDATION Fieldbus HSE signal and IEC-61131 variable connections in the Connections tab of CI860, using the DS65 and DS66 data types.



In addition to communication I/O channels, there are 10 channels containing extended status information from the CI860 unit, and the UnitStatus channel. See figure below.

IW1.6500	No of HSE publisher	dint	
IW1.6501	No of HSE subscriber	dint	
IW1.6502	Reserved 1	dint	
IW1.6503	No of send failed	dint	
IW1.6504	No of exp. stall count	dint	
IW1.6505	Average FF load	dint	
IW1.6506	UDP received w/o processing	dint	
IW1.6507	Reserved 2	dint	
IW1.6508	Reserved 3	dint	
IW1.6509	Reserved 4	dint	
IW1.6510	UnitStatus	dint	

INSUM Communication Library

The INSUM Communication library (INSUMCommLib) contains function block types and data types for INSUM (Integrated System for User-optimized Motor control) communication.

INSUM is a system for protection and control of motors and switchgear. AC 800M controllers communicate with the INSUM system via TCP/IP, using the communication interface CI857.

Usage and Status Information for INSUMConnect Function Block

To establish connection using INSUMConnect, set the value of the **En_C** parameter to 'true', and specify the remote system with the CIPos and GWPos parameters. A reference to the connection is inserted into the **Id** parameter so that this parameter can be used by other function blocks communicating via the same connection (for example, INSUMReceive and INSUMWrite function blocks).

The execution status of INSUMConnect is presented via the following parameters:

- Valid
 - A "bool" parameter that indicates if the connection is working (true) or not (false)
- Error
 - A "bool" parameter that is true during one execution cycle, after the detection of an error.
- Status

A "dint" parameter that gives a value about the execution status of the function block. A negative value means an error.

- **MsgStatus**

A "INSUMGWMsgStatus" structure that contains status information about the connection. This information is received from the gateway.

- **GWStatus**

A "INSUMGWStatus" structure that contains status information about the gateway. This information is collected by other means that just handles the connection (for example, the supervision of the CI857 module).

Usage and Status Information for INSUMReceive Function Block

To activate cyclic reading of data through INSUMReceive, set the value of **EN_R** parameter to 'true', and connect the **Id** parameter of INSUMReceive to the **Id** parameter of an INSUMConnect function block.

The execution status of INSUMReceive is presented via the following parameters, apart from the common parameters like Valid, Error, and Status (which are described for INSUMConnect):

- **Ndr**

A "bool" parameter that is set to 'True' during one execution cycle, after the new data is received through the **Rd** parameter or any of the status parameters.

- **MsgStatus**

An "INSUMDeviceMsgStatus" structure that contains status information about the Network Variable subscription created by this INSUMReceive block. This information is received from the gateway.

- **DeviceStatus**

A "dint" value that contains status information about the INSUM device from which the INSUMReceive block receives data. This information is received from the Field Device List in the gateway.

Usage and Status Information for INSUMWrite Function Block

To run the write operation through INSUMWrite, set the value of the **Req** parameter to 'true', and connect the **Id** parameter of INSUMWrite to the **Id** parameter of an INSUMConnect function block.

The execution status of INSUMReceive is presented via the GWMsgStatus parameter, apart from the common parameters like Error and Status (which are described for INSUMConnect). The GWMsgStatus is a "dint" field that contains status information about how the write operation is executed. This information is received from the gateway.

The INSUM system consists of devices that are connected via a LonWorks network. There are different device types for different types of equipment that can be controlled and supervised. The device type used for motor control is called a Motor Control Unit (MCU). The MCU is located in the motor starter module.

Network Variables in Motor Control Units (MCU)

The table shows Network Variables that are defined in the INSUM Motor Control Unit.

Function/Object in MCU	NV name in MCU	Dir.	Description
Current Measurement	nvoCurrRep	In	Current information: A, % and Earth current
TOL (Thermal overload)	nvoCalcProcVal	In	Thermal capacity: % to Thermal Overload
	nvoTimeToTrip	In	Estimate of time until the motor will trip due to thermal overload based on the current load.

Function/Object in MCU	NV name in MCU	Dir.	Description
Motor Control	nvoTimeToReset	In	Remaining time until it is possible to reset the MCU after a thermal overload trip.
	nviDesState	Out	Commands: Start, Stop etc
	nvoCumRunT	In	Cumulated run hours
	nvoMotorStateExt	In	Motor status: Running, Stopped, Alarm etc
Contactors 1	nvoOpCount1	In	Number of switch cycles for contactor 1.
Contactors 2	nvoOpCount2	In	Number of switch cycles for contactor 2.
Contactors 3	nvoOpCount3	In	Number of switch cycles for contactor 3.
Control Access	nviCAPass	Out	Control access commands: Local/Remote control of the device
	nvoActualCA1	In	Feedback of Control access commands
Node	nvoAlarmReport	In	Alarmreport with Warning- and Trip information
Voltage Measurement	nvoVoltRep	In	Phase voltages and frequency
Power Measurement	nvoPowRep	In	Motor power: Active power, reactive power and power factor
General Purpose I/O	nviGpOut1	Out	General Purpose Output 1
	nvoGpOut1Fb	In	Feedback of General Purpose Output 1
	nviGpOut2	Out	General Purpose Output 2
	nvoGpOut2Fb	In	Feedback of General Purpose Output 2
	nvoGpIn1	In	General Purpose Input 1
	nvoGpIn2	In	General Purpose Input 2

Network Variables in Circuit Breakers

The table shows Network Variables that are defined in the INSUM Circuit Breakers.

Function/Object in Circuit Breaker	NV name in Circuit Breaker	Dir.	Description
Node	nvoNodeAlarmRep	In	Alarm report with Warning- and Trip information
	nviNodeCommand	Out	Commands: Open, Close etc
	nvoNodeStatusRep	In	Circuit Breaker Status: Closed, Open, Alarm etc
RMS Current	nvoAmpsCurrRep	In	Current information: A, % and Earth current
Control Access	nviCAPass	Out	Control access commands: Local/Remote control of the device
	nvoCAOwner	In	Feedback of Control access commands

MB300 Communication Library

The MB300 Communication library (MB300CommLib) contains function block types for MB300 communication. The MasterBus 300 (MB 300) protocol can be used with AC 800M and AC 400. The CI855 communication interface unit for AC 800M is used to connect to AC 400 controllers via MasterBus 300.

Dataset communication between controllers connected to MasterBus 300 is handled by three function blocks. A dataset consists of an address part and up to 24 elements (32-bit values). Values can be a 32-bit integer, a 16-bit integer, a real or 32 booleans.

Each CI855 unit behaves as a unique node on the MasterBus 300 network it is connected to, and has to be configured accordingly in the Control Builder hardware tree.

MMS Communication Library

The MMS Communication library (MMSCCommLib) contains MMS data types, function block types and control module types for establishing communication with systems using the MMS protocol. MMS (Manufacturing Message Specification) is used as a common application layer protocol. MMS defines communication messages transferred between units, and has been specifically designed for industrial applications.



SIL certified communication is supported, see the *AC 800M Communication Protocols (3BSE035982*)*, and [SIL Certified Communication](#) on page 358.

MMS is the base protocol in Control Network. All communication between Control Builders/OPC Servers and controllers uses MMS, for example, project download, firmware download and online communication. Alarm and event handling also uses MMS.

If the MMS Communication library is used, the communication between controllers can be defined using access variables and function block types and/or control module types from the MMS Communication library.



It is also possible to define the communication between controllers without using MMSCCommLib, by using the IAC feature and communication variables.

For more information on MMS communication, see the *AC 800M Communication Protocols (3BSE035982*)*.

SIL Certified Communication (MMS)

The MMS communication library provides data types, function block types and control module types for safe communication with other controllers or between applications running in the same controller.

A number of the MMS communication function blocks and control modules are SIL marked, see [SIL Certified Application](#) in the *System 800xA Control AC 800M Getting Started (3BSE041880*)*.



For more information on safe communication, see [SIL Certified Communication](#) on page 358.

MODBUS RTU Communication Library

The MODBUS RTU Communication library (ModBusCommLib) contains data types and function block types for communication via the MODBUS protocol.

MODBUS can be used for point-to-point or multidrop communication. Communication takes place serially and asynchronously, based on the master/slave principle, and in half duplex. MODBUS slave communication is not supported, only master communication.

MODBUS TCP Communication Library

The MODBUS TCP communication library (ModBusTCPCommLib) contains function blocks types supporting the MODBUS TCP protocol. The types are used for MODBUS TCP communication through Ethernet ports on CI867.

MODBUS is a request response protocol and offers services specified by function codes and supports both master and slave functionality.

The master functionality provides the possibility to access registers and coils in other MODBUS TCP devices for both write and read operations. It is also possible for masters to retrieve status/diagnostic information from the slaves.

The slave functionality provides the possibility for other devices to access Access Variables. Both read and write operations are possible.

Modem Communication Library

The Modem Communication library (ModemCommLib) contains function block types used for serial communication over a modem. To use a modem connection, the modem must be configured to a serial (Com) port and the COMLI protocol must be added and configured (for more information, see Control Builder online help).

For more information about modem communication, see also the *AC 800M Communication Protocols (3BSE035982*)*.

Siemens S3964 Communication Library

The Siemens S3964 Communication library (S3964CommLib) contains function block types to establish communication with a system supporting the Siemens 3964R protocol.

Siemens 3964R is a point-to-point protocol, which means that only one Siemens system can be connected to each channel. The Siemens system requires an Interpreter RK 512 unit.

SattBus Communication Library

The SattBus Communication library (SattBusCommLib) contains function block types supporting SattBus. The types are used to communicate through Ethernet, using the SattBus name-oriented model.

SattBus is only available for TCP/IP on Ethernet.

MTM Communication Library

The MOD5-to-MOD5 communication library, MTMCommLib, provides function blocks to implement variable communication client in MOD5 controller to AC 800M communication.

The MTMCommLib contains function block types.

The MOD5-to-MOD5 (MTM) protocol consists of request and response messages that are exchanged each second.

The requests sent to other connected systems are determined by the control application. The response sent at each second is determined by the requests received at the previous second from other connected systems. The application programmer accesses the protocol functions through standard function blocks.

The library uses the functions blocks MTMConnect, MTMReadCyc, MTMDefCyc, and MTMDefERCyc to translate the request and to answer the MOD5 commands.

Serial Communication Library

The Serial Communication library (SerialCommLib) contains function block types for communication with external devices (for example printers, terminals, scanner pens) via serial channels with user-defined protocols. You can write an application which controls the characters sent and checks whether the correct answer is received, using serial channel handling function blocks.

Some of the function block types in SerialCommLib are certified SILx Restricted. They are allowed to be used in SIL classified applications, but the communicated data can not be used for safety critical functions without adding a safety layer as described in [SIL Certified Communication](#) on page 358.

The following use cases help in understanding the libraries used in serial communication.

Establishing a valid connection for serial communication

The prerequisites to establish a valid serial port connection for reading data from a physical device or writing data to a physical device are:

- Download the 61131-3 application that contains SerialCommLib and SerialHWLib.
- Instantiate the SerialConnect function block.

After the connection is established, the protocol is configured with the default behavior (read and write messages ended by an EOM (End of Message)). The default behavior is described by the parameters like En_C, Channel, Partner, Valid, Error, Status, and ID, which are present in the function block.

If a malfunction of the connected CI853 communication module is detected, the module can be replaced by a new one, and the connection to the serial port is automatically reestablished.

Adding a CRC calculation to a message in serial communication

The prerequisites to add a CRC calculation to a message are:

- Establish a valid connection to the serial port.
- Instantiate the SerialSetup function block in the 61131-3 application.

After the CRC calculation is added, the settings of CRC remain intact even after a disconnect operation.

Enabling basic listening for serial communication

The prerequisites to enable the basic listening of serial communication data from a device are:

- Establish a valid connection to the serial port.
- Instantiate the `SerialListen` function block in the 61131-3 application.

The input parameters for basic listening are message length, end of message, and number of trailing characters.

After the basic listening is enabled, the string message (which is the output seen in the `Rd` parameter) is received by the input device.

Enabling basic writing of serial communication data

The prerequisites to enable the basic writing of serial communication data to a device are:

- Establish a valid connection to the serial port.
- Instantiate the `SerialWrite` function block in the 61131-3 application.

After the basic writing is enabled, the output is an acknowledgment from the `Sd` parameter.

Example (Buffer handling)

A `SerialListen` function block is set up to read a specified message length of for example 5 characters (`MsgLength = 5`).

While the `Enable` parameter has the value `True` and the buffer contains characters the `Ndr` parameter will be `True` and 5 characters at a time will be passed to the `Rd` parameter.

If an incoming message "012345678901234" has been received with a size of 15 characters (3x5) and is stored in the buffer the following will occur:

First scan: `Rd = 01234 (012345678901234)`, `Buffer = 5678901234`

Second scan: `Rd = 56789 (012345678901234)`, `Buffer = 01234`

Third scan: $Rd=$ 01234 (012345678901234), Buffer is empty

There will be no fourth scan since the buffer is empty.

If the message length is not a multiple of the *MsgLength* parameter the buffer will keep the remaining characters until the number of characters in the buffer again is greater than or equal to the *MsgLength* parameter value.

If an incoming message "0123456789012" has been received with a size of 13 characters ($2 \times 5 + 3$) and is stored in the buffer the following will occur:

First scan: $Rd =$ 01234 (0123456789012), Buffer = 56789012

Second scan: $Rd =$ 56789 (0123456789012), Buffer = 012

There will be no third scan as the buffer does not contain at least 5 characters. The buffer will retain these values until additional characters are added to the buffer and it once again equals, or exceeds, 5 characters in length. At that time, the first 5 characters will be passed to the *Rd* parameter.

By setting the *En_C* parameter of the *SerialConnect* function block to value False (disconnecting), the buffer of the serial channel will be cleared.

TCP and UDP Communication Libraries

The UDP communication library (UDPCommLib) and TCP communication library (TCPCommLib) library provide function blocks that can be used to implement protocols running on UDP or TCP.

The typical usage of these libraries is when the controller needs to communicate with external equipment. Some examples are:

- Communication with different road-infrastructure network nodes such as variable speed signs, traffic direction and information signs.
- Vision cameras – Many vision cameras implement the Telnet protocol (ASCII TCP communication over standard port number 23).
- Information server – The controller may function as both client and server on the network. Example of server use is a SCADA application where a supervisory system connects to different servers and collects information periodically.

UDP communication does not establish a connection prior to sending/receiving. The UDPCommLib supports broadcasting with which one node can send a message that is received by many nodes.

TCP is a point-to-point connection. Using TCPCommLib also ensures that the messages are delivered without loss.

The only hardware configuration that is needed in Control Builder is to attach the UDPProtocol and TCPProtocol hardware types (from UDPHwLib and TCPHwLib respectively) under the **IP** position in the AC 800M controller hardware tree.

Generic IO Communication Library

The Generic IO Communication library (IOCommLib) contains the function block types IOConnect, IORead and IOWrite for acyclic communication. These function block types are generic to be used by different communication protocols. PROFINET IO with CI871 is the first communication protocol that uses it.

Supported Protocols

Table 25 lists all supported protocols.

Table 25. Protocols supported by Control Builder

Protocol	Port/Interface
MMS on Ethernet	CN1, CN2 (TP830)
MMS on RS-232C (PPP)	COM3 (TP830), CI853
MasterBus 300	CI855
SattBus on TCP/IP	CN1 (TP830)
COMLI ⁽¹⁾	COM3 (TP830), CI853
Siemens 3964R ⁽²⁾	COM3 (TP830), CI853
MODBUS RTU ⁽¹⁾	COM3 (TP830), CI853
MODBUS TCP on Ethernet ⁽¹⁾	Ch1, Ch2, CI867
IEC 61850	Ch1, CI868
FOUNDATION Fieldbus HSE	CI860
PROFIBUS DP	CI854
DriveBus	CI858
INSUM	CI857
MOD5-to-MOD5	CI872
AF 100	CI869
PROFINET IO	CI871
EtherNet/IP	Ch1, CI873
UDP	Ch1, Ch2
TCP	Ch1, Ch2

(1) Both master and slave

(2) Master only

For more information on supported protocols, see the *AC 800M Communication Protocols (3BSE035982*)*.

Control Network

Control Network is a private IP network domain especially designed for industrial applications. This means that all communication handling will be the same, regardless of network type or connected devices. Control Network is scalable from a very small network with a few nodes to a large network containing a number of network areas with hundreds of addressable nodes (there may be other restrictions such as controller performance).

Control Network uses the MMS communication protocol on Ethernet and/or RS-232C to link workstations to controllers. In order to support Control Network on RS-232C links, the Point-to-Point Protocol (PPP) is used.



For information on time stamps and clock synchronization within Control Network, see the *AC 800M Communication Protocols (3BSE035982*)*. Time synchronization is also briefly described in [Section 2, Alarm and Event Handling](#).

Control Network, as well as other protocols and fieldbuses, is configured using Control Builder (via the Project Explorer interface). Control Network settings are specified in the parameter lists, accessed by right-clicking CPUs, Ethernet ports and/or PPP connections.



The address of controller Ethernet ports should in some cases be set using the IPConfig tool. See the *System 800xA Control AC 800M Getting Started (3BSE041880*)*.



For information on communication parameter settings, see Control Builder online help for the object in question. Select the object in Project Explorer, then press F1 to display the corresponding online help topic.

Network Redundancy

The *Redundant Network Routing Protocol (RNRP)*, developed by ABB, handles alternative paths between nodes and automatically adapts to topology changes.

For more information on redundancy and RNRP, see the *System 800xA Network Configuration (3BSE034463*)*.

Statistics and Information on Communication

Statistics concerning all MMS communication and Inter Application Communication (IAC) in a system are displayed in the Remote System dialog. Information can be viewed at any engineering station that is connected to the network, by selecting **Tools > Maintenance > Remote System**, followed by **Show Remote Systems**.

You can get the following MMS-related information:

- **Tools > Maintenance > Remote System > Show MMS Variables** shows which MMS variables are present in the selected remote system
- **Tools > Maintenance > Remote System > Show MMS Connections** shows all connections, including information on the type of connection, the destination system, and a number of statistics.

The information regarding IAC (using communication variables) can be accessed from **Tools > Maintenance > Remote System > Show Diagnostics for Communication Variables**. See [Diagnostics for Communication Variables](#) on page 471.

There is also a function block type System Diagnostics that is stored in the Basic library. This function block will (among other things) show Communication variables, IAC, and Ethernet statistics.



For more information on the contents of the Remote System dialog and the System Diagnostics function block type, see Control Builder online help.

Variable Communication

Communication between applications uses access variables and communication variables.

Access Variables

Access variables are defined in the access variable editor, which is displayed by double-clicking Access Variables in the Controllers folder. The access variable editor can also be displayed from the application editor, by double-clicking an access variable field in the Access Variables column.

Access variables can use the MMS, COMLI, MODBUS TCP and SattBus protocols. MMS and SattBus variables are declared in the Access Variable Editor under the corresponding tab, COMLI and MODBUS TCP variables under the Address tab.

Paths to local variables are given using the syntax

```
ApplicationName.ProgramName.FunctionblockName.VariableName
```

Communication Variables

Communication variables are used for cyclic communication between top level diagrams, programs, and top level single control modules. These objects can exist in the same application, the same controller, or in another controller. The name of the communication variable must be unique on the network to resolve the IP-address during compilation.

Communication variables behave differently depending on where the *in* and *out* variables are placed:

- *In* and *out* variables are in the same application and connected to the same IEC 61131-3 task
 - In this case, the *in* and *out* variable represents the same physical memory location; hence no communication is setup.

- *In* and *out* variables are in the same application, but connected to different IEC 61131-3 tasks; or the *in* and *out* variables are in different applications in the same controller
 - In this case, fast data copying is performed at each 61131-3 task scan for the *in* variable. This type of IAC is called internal IAC, where the data is copied between different memory locations, and this does not involve any real communication. This is controlled by the task time, hence no external communication is setup.
 - If the *in* variable is defined in a SIL3 application and the *out* variable is defined in a lower SIL application, communication is driven by considering the *in* variable as external variable. This results in external IAC, and the client receives new values based on the configured interval time. This case is different from the normal internal IAC.
- *In* and *out* variables are in different applications in different controllers (external communication)

In this case, the actual external IAC occurs, and the client receives new values based on the configured interval time. The protocol used is IAC_MMS, which is based on User Datagram Protocol (UDP).

Five different interval time categories are used and these are configured on the IAC_MMS hardware unit in Control Builder.

Communication variables are declared in the editor for diagrams, programs and top level single control modules in Control Builder.



The communication using the declared communication variables happens only if the **IP** and **IAC MMS** hardware types are inserted under the controller in the hardware tree in Control Builder.



For more information about variable communication, see [Variables and Parameters](#) on page 81.



Communication with SIL-certified applications and AC 800M High Integrity controllers is restricted, and must use Inter Application Communication (IAC) (using communication variables) or SIL certified MMS communication function blocks. For more information on SIL communication restrictions, see the *System 800xA Safety AC 800M High Integrity Safety Manual* (3BNP004865*), See also [SIL Certified Communication](#) on page 358.

StartAddr

All read and write function blocks have a StartAddr parameter. The StartAddr identifies the first requested variable in the remote system.

Set a prefix and a start address via the StartAdr parameter. This sets the access variable which identifies the memory area in the remote system from which data is to be read or to which it is to be written.

For further information regarding memory addressing: see IEC 61131-3 Variable Representation for IEC 61131-3 direct addressing and Access Variable Syntax for direct addressing.

Example 1

You can read 16 bits from a subsystem, starting from the decimal address 64 (octal address 100), as follows.

Connect a structured variable declared with 16 Boolean components to the Rd[1] parameter in the COMLIRead function block. Then set the StartAddr parameter to:

Table 26. StartAddr parameter setting (16 bits)

Protocol	IEC 61131-3 Direct Addressing	Direct Addressing (Octal, 8# only)
MODBUS RTU/ MODBUS TCP	%IX8#100 (input) %QX8#100 (output) %IX10#64 (input) %QX10#64 (output) %IX16#40 (input) %QX16#40 (output)	Not supported
COMLI	%MX8#100 %MX10#64 %MX16#40	%X100 or X100
Siemens 3964R	%MX8#100 %MX10#64 %MX16#40	%X100 or X100

Text in bold face indicates the most commonly used values.



If you exclude the base from the format it is assumed to be base 10. For example, %MX64 is interpreted as %MX10#64.

Example 2

You can read a Register 45 from a subsystem, starting from the decimal address 45, as follows:

Connect a structured variable declared with 16 Boolean components to the Rd[1] parameter in the COMLIRead function block. Then set the StartAddr parameter to:

Table 27. StartAddr parameter setting (Register 45)

Protocol	IEC 61131-3 Direct Addressing	Direct Addressing (Octal, 8# only)
MODBUS RTU/ MODBUS TCP	%MW8#55 %IW8#55 (input) %QW8#55 (output) %MW10#45 %IW10#45 (input) %QW10#45 (output) %MW16#2D %IW16#2D (input) %QW16#2D (output)	Not supported
COMLI	%MW8#55 %MW10#45 %MW16#2D	%R45 or R45
Siemens 3964R	%MW8#55 %MW10#45 %MW16#2D	%R45 or R45

Text in bold face indicates the most commonly used values.



If you exclude the base from the format it is assumed to be base 10. For example, %MW45 is interpreted as %MW10#45.

Reading/Sending Data

The communication libraries contain all types you need to set up communication for the supported protocols. For most protocols, there are three main types:



Due to variations between various protocols, the name of individual types and parameters may vary slightly between the different communication libraries. However, the communication principles are still the same.



Communication function blocks should not be called more than once per scan. Exceptions to this are stated explicitly in the corresponding online help. Do *not* call communication function blocks in SFC, in IF statements, in CASE statements, etc.

- **Connect Types**

Connect types are used to initiate a communication channel and establish a connection to a remote system with a unique node address in a network. Connect types are used to open a communication channel. The identity of the opened channel is communicated to the Read and Write types via an identity parameter (the exact name of this parameter varies between protocols). For example, MMSConnect is used by MMSRead and MMSWrite.

A connection is established when an enable parameter is set to true. This means that a communication channel can be opened whenever needed. The identity of the system to which a connection has been established is communicated to the corresponding read and write types via an *Id* parameter.

Connect types have a built-in continuous supervisory function, which detects if communication is interrupted after connection has been established.



Normally you use the VarName parameter in MMSRead function block for reading different variables online. However, all the MMSRead4* function blocks are a little bit different. For example, you cannot change the VarName parameter directly in Online for these MMSRead4* function blocks. Instead, after changing the VarName parameter (for reading another variable), you must also set the parameter EN_C in the MMSConnect function block to false (one scan) and then back to true again before the new variable can be read.



The MMSWrite is used for communication between applications.

Communication between applications residing in different controllers is called external, and is asynchronous. Communication between applications within the same controller is called internal. The internal copy is synchronous or asynchronous based on the amount of data copied.

The basic algorithm is based on how much data that can be copied synchronously without disturbing the execution of 1131 tasks, that is the task latency must be less than 2 ms. The total amount of data to be copied is about 800 bytes of variable data. If the total of 800 bytes is exceeded the internal copy may affect the task latency negatively and therefore the internal copy is executed asynchronously.

- **Read Types**
Read types read data (often an access variable) from a target system. The source system (the communication channel) is indicated by the *Id* parameter, which is passed from the corresponding connect function block or control module.
- **Write Types**
Write types write data to a target system. The target system (the communication channel) is indicated by the *Id* parameter, which is passed from the corresponding connect function block or control module.

For some protocols, there are also additional types, such as types for cyclic reading of data, data conversion, download of measuring ranges, etc.

Connection Methods

Function blocks from the communication libraries are used to read and write variables from a remote system:

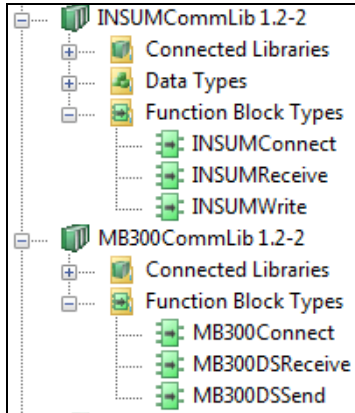


Figure 147. Function blocks in the communication libraries.

In the application program, a common *Connect* function block is used in a client (master) to establish connection to a server (slave). The function blocks *Read* and *Write* can then be used repeatedly. Refer to online help for a description of the parameters concerned. Variables to be accessed must be declared in the server Access variable editor.

To display the editor, right-click the Access Variables object and select Editor.

Example 1:

Controller 2 (client) connects to Controller 1 (server) by means of a *Connect* function block. Refer to online help for a description of how *Partner* and *Channel* are specified for different communication protocols. *Read* and *Write* function blocks with the same identity (ID) as the Connect block can then be used repeatedly.

As an example, Controller 2 has a Read function block in its application program that sends a Read request to Controller 1 for an access variable named %R100. This name must exist in the access variable list in Controller 1, which then reads the value of Program1.A (%R100) and sends it to Controller 2. The value is then written to the application variable named in *Rd*.

In the same way, the value of a variable in the Controller 1 access variable list can be changed by means of a Write function block in Controller 2.

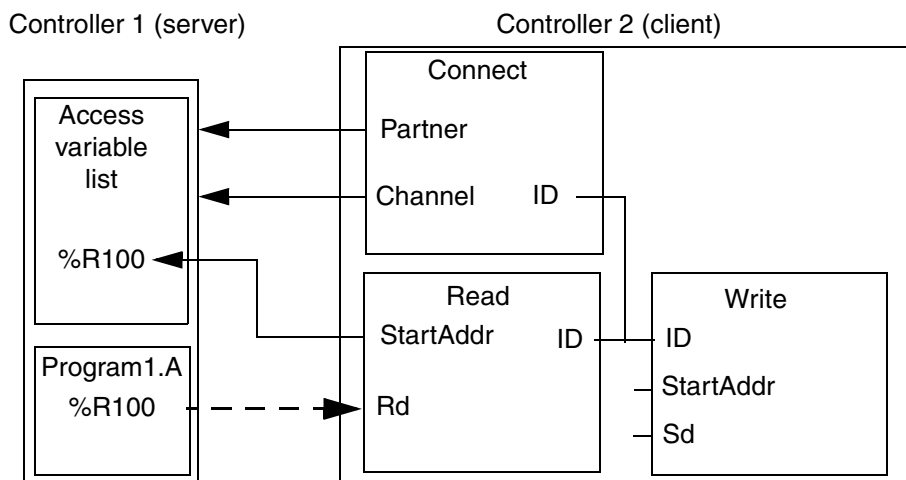


Figure 148. Variable read by controller 2, from controller 1.

The function blocks *ReadCyc* and *WriteCyc* perform in a similar manner, but are used to cyclically read or write to/from a server system with the interval specified by the *SupTime* parameter.

Example 2:

Write and read requests are triggered by the *Req* parameter being set to True after having been False for at least one scan. This problem can be avoided if two function blocks are executed, one after the other. In this way, a request is always outstanding. Additional requests triggered by the *Req* parameter will be ignored by the function block, until the *Done* (or *Ndr*) parameter has become True.

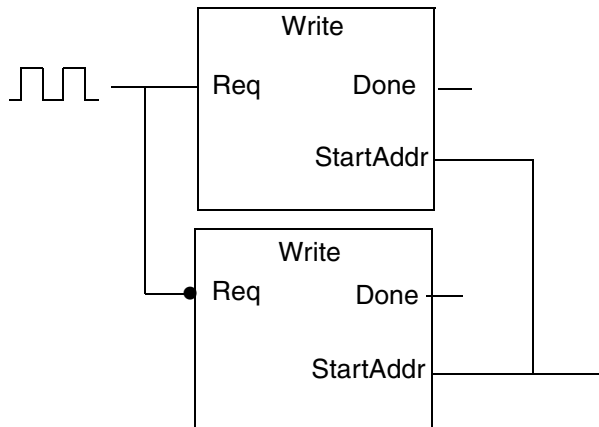


Figure 149. Resetting the Req parameter using two function blocks.

Communication Concepts

When setting up communication with external devices and other controllers, it is also important to be familiar with the following:

- The client/server concept (master/slave), see [Client/Server Communication](#) on page 352,
- The publisher/subscriber (also called subscriber/provider) concept, see [Publisher/Subscriber Communication](#) on page 353.
- There is also the choice between cyclic and asynchronous communication, see [Cyclic vs. Asynchronous Communication](#) on page 354.

Client/Server Communication

The main principle of client/server communication is the following:

- The client is the active party, which requests (reads) data from the server.
- The server is a passive provider of information that simply answers to requests from the clients.



Client/server communication could also be described as master/slave communication. In that case, the client is the master, and the server is the slave.

Figure 150 shows the principle.

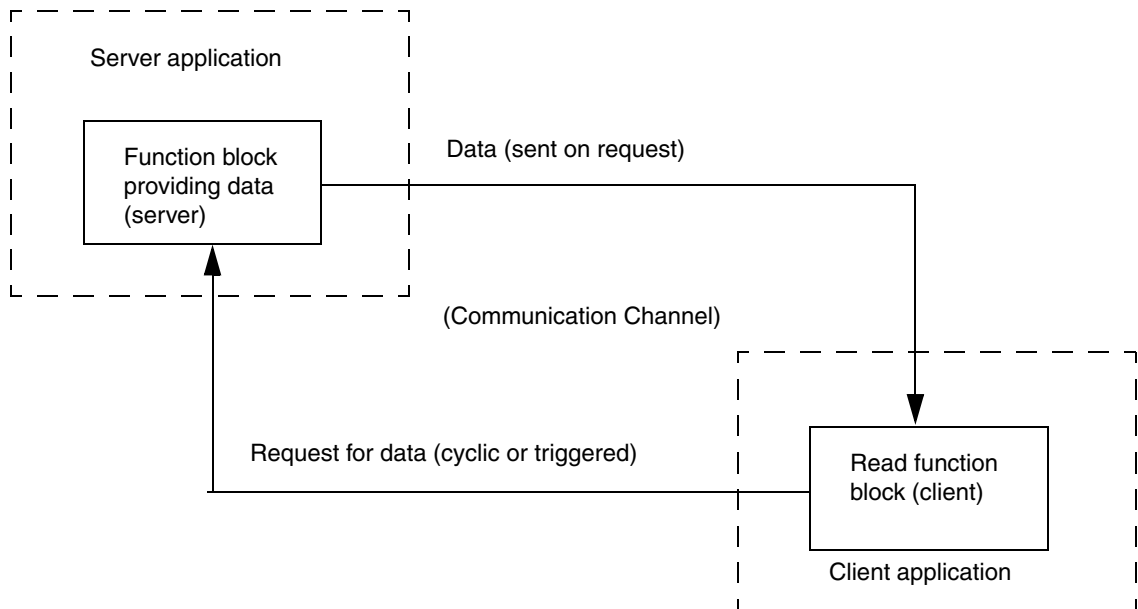


Figure 150. Client/server principle. The client reads data from the server. The server sends data to the client when requested.

Publisher/Subscriber Communication

The main principle of publisher/subscriber communication is the following:

- The publisher publishes (the publisher is also known as the provider) data cyclically, in a pre-determined location.
- The subscriber is a consumer of information, which subscribes to published data.

Figure 151 shows the publisher/subscriber principle.

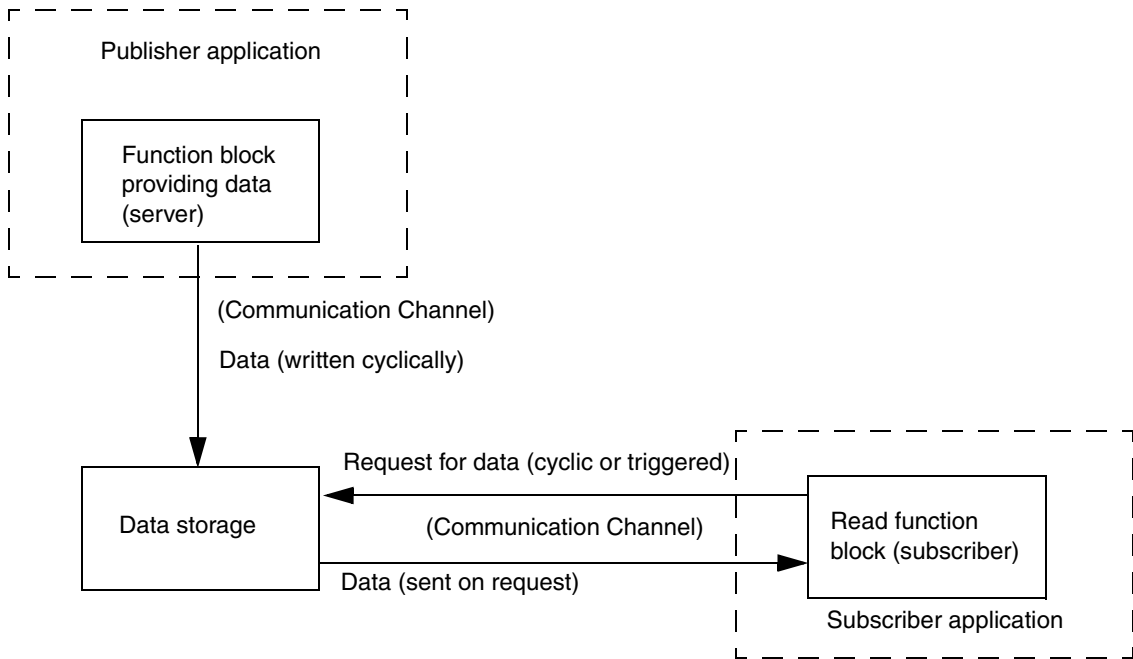


Figure 151. Publisher/subscriber principle. The publisher publishes data to a pre-defined location, which is read by the subscriber.

Cyclic vs. Asynchronous Communication

An important decision when setting up communication is whether communication should be cyclic, that is, take place regularly, with a certain time interval, or asynchronous, that is, take place when triggered by a certain event or condition.

Which method to use depends on things such as:

- How much does the execution of communication code affect performance?
- How often can a value be expected to change?

- How important is it that a change in a certain value is communicated immediately?



For more information about communication, performance and design, see *System 800xA Control AC 800M Planning (3BSE043732*)*.



For information on how to make part of your code execute with a different interval, see [Control the Execution of Individual Objects](#) on page 113.

Fieldbus Communication

Fieldbuses offer communication on a dedicated bus, using a special fieldbus communication protocol. Fieldbus devices often contain distributed code, which means that they need to be set up not only from Control Builder, but also using a fieldbus-specific configuration tool.



For detailed information on how to configure the fieldbuses, refer to the corresponding, fieldbus-specific documentation. For detailed information on how to configure communication with fieldbus devices, see the corresponding Control Builder online help.



Fieldbus communication requires separate licenses.

PROFIBUS DP

PROFIBUS (PROcess FIEld BUS) is a fieldbus standard, especially designed for communication between systems and process objects. This protocol is open and vendor independent. It is based on the standard EN 50 170. With PROFIBUS, devices from different manufacturers can communicate without special interface adjustments. PROFIBUS can be used for both high speed, time critical transmission and extensive, complex communication tasks.

PROFIBUS has defined the three types of protocol: PROFIBUS FMS, DP and PA. With AC 800M access to PROFIBUS DP and PA is supported.

PROFIBUS DP is connected to the controller via the CI854/CI854A communication interface unit. The connection to PROFIBUS PA can be established by use of the Linking Device LD 800P that links between PROFIBUS DP and PROFIBUS PA.

The original version of PROFIBUS DP, designated PROFIBUS DP-V0, has been expanded to include version DP-V1 and DP-V2. With CI854/CI854A support for DP-V1 and the acyclic services (toolrouting) is given. In addition CI854/CI854A supports line and slave redundancy and CI854A supports master redundancy as well.

The PROFIBUS DP-V0 configuration and parameter data for slave devices are engineered in Control Builder and downloaded via CI854/CI854A.

PROFIBUS slave types are usually supplied with a *.gsd file. This file describes the properties of the slave type. The *.gsd file must be converted with the Device Import Wizard, in order to be used in the project.

PROFINET IO

PROFINET is a manufacturer-independent Fieldbus standard for applications in manufacturing and process automation. PROFINET technology is described in fixed terms in IEC 61158 and IEC 61784 as an international standard.

PROFINET IO uses Ethernet communication to integrate simple distributed I/O and time-critical applications.

PROFINET IO describes a device model oriented to the PROFIBUS framework, which consists of places of insertion (slots) and groups of I/O channels (subslots). The technical characteristics of the field devices are described by the General Station Description (GSD) on an XML basis. The PROFINET IO engineering is performed in a way familiar to PROFIBUS. The distributed field devices are assigned to the controllers during configuration.

The PROFINET IO is interfaced to the IEC 61131 controller AC 800M, using the PROFINET IO module CI871.

PROFINET IO is based on IEEE 802.3. PROFINET IO uses Ethernet, TCP, UDP, and IP as the basis for communication. It is designed to work with other IP-based protocols on the same network.

The transmission of time-critical process data within the production facility, occurs in the Real-Time (RT) channel.

DriveBus

The DriveBus protocol is used to communicate with ABB Drives and ABB Special I/O units. DriveBus is connected to the controller via a CI858 communication interface unit.

Advant Fieldbus 100

Advant Fieldbus 100 (AF 100) is a high performance fieldbus, which is used for:

- Communication between Advant Controllers.
- Communication between Advant Controllers and S800 I/O Stations, AC 800M controllers, and so on.

Advant Fieldbus 100 supports three transmission media:

- Twisted pair (Twp)
- Coaxial (RG59 and RG11)
- Optical media.

An AF 100 bus can be built up with all the three media, where a part of one kind of media is a specific segment.

The CI869 communication interface that is attached to the AC 800M controller provides connectivity to other AC 800M, AC 160 or connectivity server over AF 100. An AC 800M controller with the communication interface CI869 behaves as an AF 100 station, receiving data from other AF 100 stations/devices. The CI869 has integrated Twisted Pair modems.

The Advant Fieldbus 100 supports two different kinds of communication:

- Process data—Dynamic data used to monitor and control a process
- Message transfer—Used for parameters, program loading, and diagnostic purposes.

HART Communication



The protocols used by the supported fieldbuses are described in detail in the *AC 800M Communication Protocols (3BSE035982*)*.

HART (Highway Addressable Remote Transducer) is an open system communication protocol that supports remote configuration and supervision of devices with HART support, via ModuleBus or via PROFIBUS DP-V1 (tool routing).



For more information on HART support, see the *AC 800M Communication Protocols (3BSE035982*)*, and the Control Builder online help. For information on how to configure tool routing, see online help for Control Builder and online help for the Tool Routing Service, which is part of the 800xA system installation.

SIL Certified Communication



SIL certified communication is only possible within and between High Integrity controllers, and only between SIL certified applications.

For SIL certified exchange of data between controllers, the following characteristics of safe peer-to-peer communication over the control network apply:

1. The transferred data is marked with the correct SIL (the application SIL or lower). For SIL certified exchange of data using IAC, a visible notification of lower SIL data being used in a higher SIL application is provided.



Data is marked with the SIL of the application, or lower. Data with a lower SIL than the application SIL is also transferred.

2. During the transfer of data, the errors originating from software, hardware, or other sources, are detected.

If SIL data is transferred over non-SIL media (this occurs when transferring data between controllers), the applications at both ends add the following diagnostics to the data transfer:

- a. Verification of contents
- b. Verification of sender and receiver address/application
- c. Verification of sequence
- d. Verification of timing (detection of “old” data)



If the *Value* components of SIL3 variables that are connected to IO are not accessed in the IEC 61131-3 code blocks or in the FD code blocks, these SIL3 variables are not copied by a background task. They are copied only by the same task as the other connected and accessed variables.

But, Non-SIL and SIL1-2 variables, whose *Value* components are not accessed in the IEC 61131-3 code blocks or in the FD code blocks, are copied by a background task.

SIL Communication Using IAC

Inter Application Communication (IAC) is the variable communication between applications. In Control Builder, IAC is implemented using communication variables, which allow cyclic communication between POU's in different applications. The communication variables can be used in the IEC 61131-3 code blocks in top level single control modules and programs, and also in the code blocks in top level diagrams.

IAC is based on the client-server concept. In the server POU, the data is copied-out through the communication variable, after the execution of the code.

In the client POU, the data is copied-in through the communication variable, before the execution of the code.

SIL IAC (IAC involving SIL applications in HI controllers) conforms to IEC 61508 and ISO 13849-1 standards.



SIL IAC fulfills all the requirements for transferring data over non-SIL media.

The IAC is configured by declaring an *out* communication variable in one POU and one or more *in* communication variables, with the same name, in another POU. The IAC POU's can exist in the same controller or in another controller (peer-to-peer).

Safe Communication with IAC

For safe communication with IAC, the following safety features need to be configured while declaring communication variables:

- Unique ID – An integer value that logically connects an *in* variable to an *out* variable. An *in* variable with a particular Unique ID can read only from an *out* variable with the same Unique ID.

- Expected SIL – Specifies the expected SIL of the server application. This is required for the client application to interpret an incoming response. The client checks that this Expected SIL matches with the SIL in the received response.
- Acknowledge Group – Specifies how communication shall be restarted after the communication goes to ISP and the ISP values get latched. For SIL1-2 or SIL3 communication, the default value of Acknowledge Group is zero, which is not allowed in a SIL application. Therefore, it is mandatory to configure the Acknowledge Group to a value (either *auto* or a specific group ID) in SIL applications. The acknowledgement is done by using the CVAckISP control module in BasicLib.

Additionally, the extracted statuses of a communication variable in a SIL application can also be used in the code blocks to control the logic (critical loop). The statuses are extracted using the function *GetCVStatus*, available in System library.

The statuses that can be extracted are:

- OPC quality of communication
- SIL of the server application
- Whether acknowledgment through a group ID (manual acknowledgment) is required for the communication variable to restart the communication after a failure
- Online Upgrade switch occurring in the server or client (in progress or not)
- Simulated server (controller is hardware simulated or Soft Controller) or not

Communication Between Applications

IAC using communication variables is possible between applications in different controllers and between applications in the same controller, in all the following cases:

- The communicating applications have the same SIL (SIL1-2 or SIL3)
- The communication happens from higher SIL (server application) to lower SIL (client application)
- The communication happens from lower SIL (server application) to higher SIL (client application)

However, the following restrictions apply for communication from lower SIL (server application) to higher SIL (client application):

- By default, the setting of Compiler Switches prevents this type of communication (it generates error). In Control Builder, right click the project name, and select **Settings > Compiler Switches**.

The two Compiler Switches that define this type of communication are:

- SIL1-2 communication variables in SIL3 applications
- Non-SIL communication variables in SIL1-3 applications

Change these settings to *Warning*, instead of *Error*, as required.

Hence, the usage of a signal with lower SIL always results in at least a warning.

- The Expected SIL value, declared for the *in* variable, must be the correct SIL of the *out* communication variable
- The *in* communication variable with a lower Expected SIL value can only be used in top level diagrams, with a graphical representation of the variable. The FD code block in the diagram displays the communication variable with a lower Expected SIL value, with a distinct color (yellow, in both Offline and Online modes). The Expected SIL value is also displayed as a label below the graphical communication variable object.

In the FD code block, it is not allowed to have a textually connected lower SIL communication variable (the variable connected using Connect dialog or Connections Editor). Any wrong usage results in compilation error.

- In the Difference Report Before Download, it is mandatory to review and accept the item, *Communication Variables with Lower SIL*, separately, at every download.



In normal case, lower SIL signals shall never be used. They may only be used in exceptional cases, and the usage must follow the restrictions stated in the Safety Manual.

SIL Communication using MMSCommLib



The MMS Communication Library (MMSCommLib) contains control modules that fulfill all the requirements for transferring data over non-SIL media.



For more information on individual MMS SIL types, see online help or select the type in Project Explorer and press F1.

A SIL communication link for transferring variable values between controllers requires basically two things. The first application (in controller A) must have some sort of a server or encoder module which can store the variable values in a secure way, before transfer. This can be done by having the control module define a structured access variable that protects the integrity of these variable values. However to do so, the access variable must contain not only the IEC61131 variables, but also necessary security measures for data transmission.

Secondly, the requesting application (in controller B) must have a client or decoder module which can send cyclically requests to the first application. This control module must decode the access variable and verify the contents against the safety measures defined. The control module will then after security checks, hand over the receiving variable values to the requested client's application code.

The MMS communication library contains six SIL 2 control modules and two SIL 3 control modules for data transfer between applications running in different controllers. These control modules may also be used for communication between applications running in the same controller.



For more information on how to use the control modules, see Control Builder online help.

SIL Communication with Function Blocks

The MMS communication library contains a number of SIL2 function blocks for data transfer between applications running in the same SIL2 certified environment. These functions may be used for communication of data between applications running in different controllers, only if both the applications are non-SIL. If one of the applications is SIL then the communication is restricted to be within one controller. However in this case the SIL of the transferred data will be degraded to SIL 0 (i.e. non-SIL).

The MMSReadHI and MMSDefHI control modules in MMSCommLib allows the user to set-up SIL3 classified peer-to-peer communication between controllers as well as between applications within the same controller. These Control Modules can be used to communicate both within the application, and between non-SIL, SIL1-2 and SIL3 applications.



The function block MMSRead4xxx can be used for communication of data between applications running in different controllers, only if both the applications are non-SIL. If one of the applications is SIL, then the communication is restricted to be within one controller.

MMS SIL communication function blocks have a parameter for input and output of the SIL level.



When using MMS SIL 2 communication function blocks, the SIL level is always degraded to SIL 0 for all data, both between controllers and between applications running in the same controller. For safe data transfer between controllers, always use the MMS SIL control modules.

How to Choose Function Block/Control Modules in MMSCommLib

MMSCommLib contains of function block types and control modules for different communication purposes. Different communication types have to be used dependent on controller type and if the communication is between non-SIL applications, SIL2 applications or between a non-SIL and SIL 2 application. See [Figure 152](#) how to choose correct function blocks/control modules.

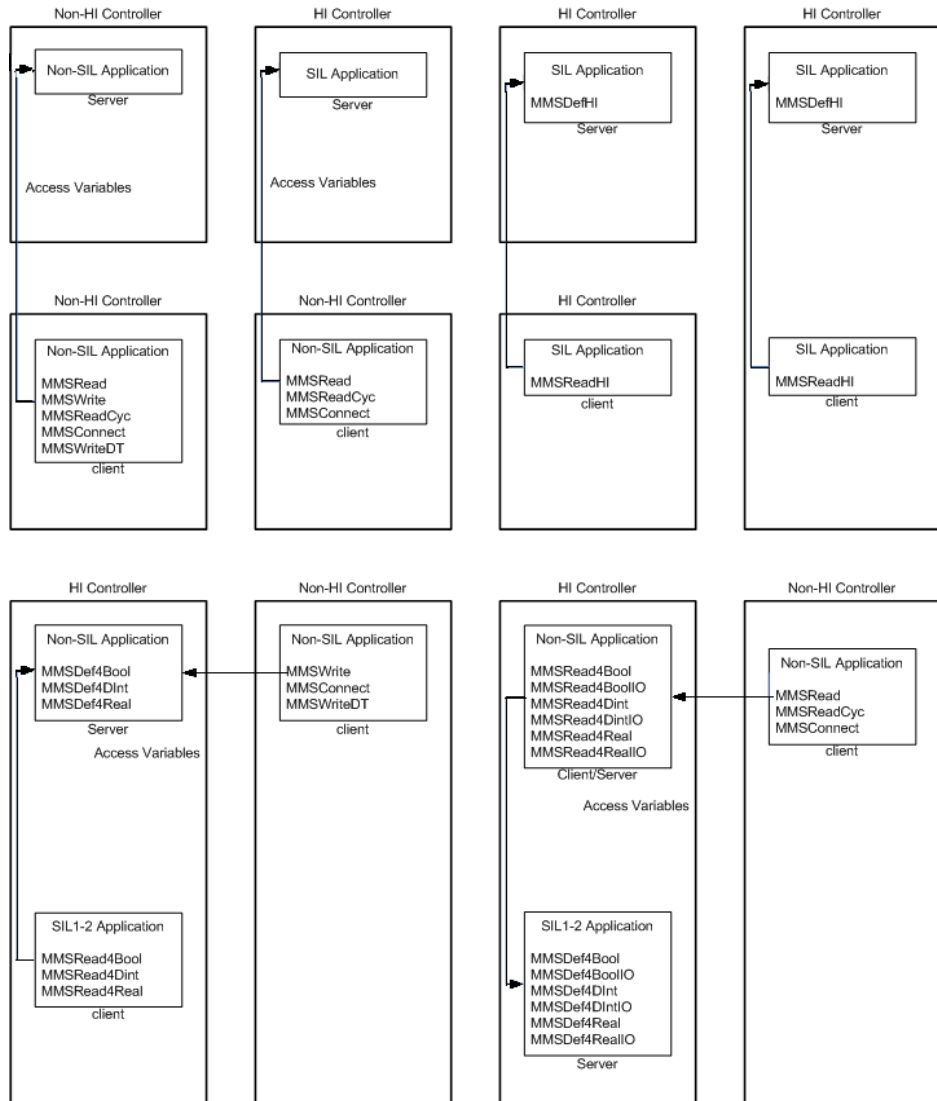


Figure 152. How to choose MMSCCommLib function blocks/control modules.

Parameter Errors (ParError)

If a parameter that is connected to a SIL certified function block or control module goes outside its range, this is indicated by the output parameter *ParError* being set to True. The *ParError* parameter is connected to interaction windows and faceplates, where a parameter error is indicated by a red triangle.



ParError also checks the values of components of structured variables, such as RealIO.



You can read more about ParError in the Extended Control Software manual.

Section 4 Online Functions

Introduction

When a controller project is in online mode and test mode, it is possible to inspect the code while running it, and interact with the code. Furthermore, you can issue operations to the controller. There are also functions to help the user to find online errors and to document the control project.



Which online changes and interactions that can be performed depends on the user permission. All user configuration is made from the Plant Explorer workplace. See [Security](#) on page 195 for more information.

The following functions are available in online mode and test mode:

- Online editors, see [Online Editors](#) on page 368.
- Dynamic display of I/O channels and forcing, see [Dynamic Display of I/O Channels and Forcing](#) on page 372.
- Scaling analog signals, see [Scaling Analog Signals](#) on page 375.
- Unit status and channel status, see [Supervising Unit Status](#) on page 375.
- Communication variable status, see [Supervising Communication Variable Status](#) on page 379.
- Hardware and task status indications, see [Status Indications](#) on page 386.
- Tasks, see [Tasks](#) on page 387.
- Interaction windows, see [Interaction Windows](#) on page 388.
- Status and error messages, see [Status and Error Messages](#) on page 390.
- Reports and analysis, see [Search and Navigation in Online and Test Mode](#) on page 391.
- Project documentation, see [Project Documentation](#) on page 395.

Online Editors

From the Project Explorer in online mode, you have access to editors similar to those in offline mode, such as the application editor, diagram editor, program editor, hardware configuration editor and function block editor. By using the online editors the code currently running in the controller(s) can be inspected. Variable values and parameters can be changed.

You can open one or several new online editor windows from the Project Explorer by double-clicking on the Program Organization Unit (POU, see [Application Types and Instances](#) on page 43) you want to view. You can also select the POU, right click, and select **Online Editor** (or **CMD Editor**).

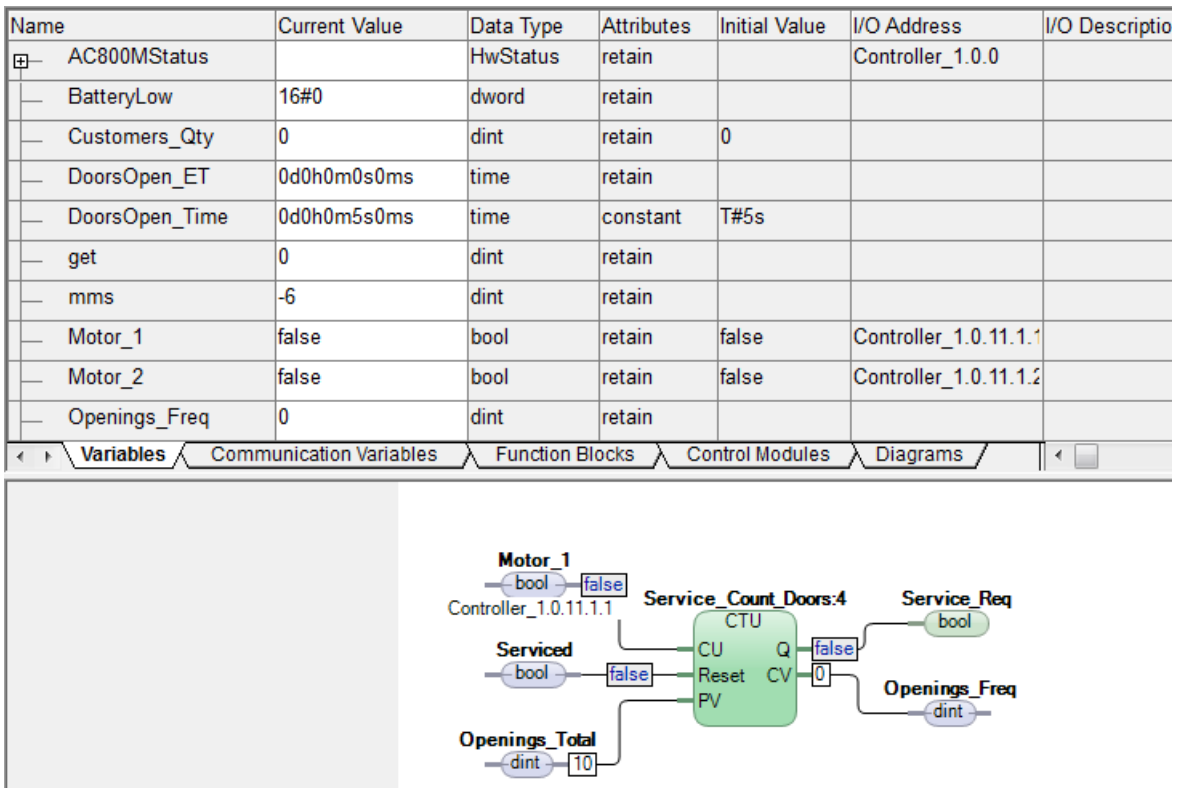



Figure 153. Part of Diagram editor in online mode


In online mode there are fewer menu entries in the menu bar than in the offline editor. **Edit** and **Insert** are not available in online mode. The options available in online menus are also somewhat different from those in offline mode. Columns in the editor that are dimmed are not accessible.

An online editor window consists of a title row, menu bar, tool bar, and a status bar at the bottom. The window is split into three panes, as follows.

- In the upper declaration pane the variables and parameters of the POU are displayed in forms that resemble Excel data sheets. Each sheet, with its tab, has a unique appearance with respect to the number of columns and their names. Select a tab to see its sheet, available columns and their names. See also Online Change of Variable Values in the online help.
- The middle code pane displays the various code blocks in the POU.
- The lower description pane displays descriptions of the types and POU's.


It is possible for the user to enter editor settings in the Setup Editor dialog, using the **Tools > Setup** menu.

From the online editor window you can activate the POU editor window using the **Tools > Edit Type** menu or the **Edit Type** button .


You can activate an online window for the POU parent via the **Tools > View Parent** menu or the **View Parent** button .



See the Control Builder online help for more information about the Setup Editor dialog, **Edit Type** and **View Parent**.

To access filter select a column in the grid and select **Tools > Filter** or **Filter** button .

From the 'Filter' dialog one can decide which rows to display or hide by selecting or deselecting criteria items. The Criteria items can also be text filtered. An icon in the column header informs shows an active filter on that column.

Alphabetical sorting of the column is possible by selecting **Tools > Sort A to Z** or **Sort Z to A**, or by clicking the Sort A to Z /Sort Z to A button in toolbar .

If column is not selected, **Sort A to Z** and **Sort Z to A** are disabled. In offline Editor, when sorting the parameter column, a warning is presented which informs that the parameter order might be changed.

Diagram Editor in Test Mode and Online Mode

In the diagram editor, in Test mode and Online mode, the current values are displayed in labels. Double-click on label, the **Set Value** dialog box opens (See Figure 154).

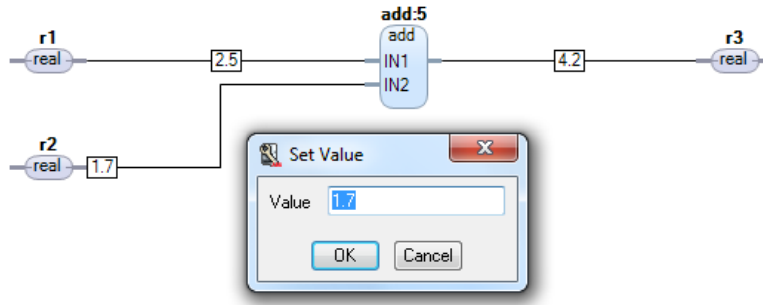


Figure 154. Set Value dialog

If the data type is bool, right click the displayed value (*true* or *false*) and set **On** or **Off** (to toggle the value).

For graphical connections of structured data type, the online value label is not shown. However, if one of the components is marked with the *displayvalue* attribute in the data type editor (see Figure 155), a label is shown in the diagram editor.

The screenshot shows the 'Data Type - Application_1.DataTypeWithDisplayValue' editor. It contains a table with the following data:

Name	Data Type	Attributes	Initial Value	ISP Value	Description
c1	real	retain			
cDisplayValue	real	displayvalue	2.6		

The 'displayvalue' attribute in the second row is circled in red.

Figure 155. Component marked as displayvalue

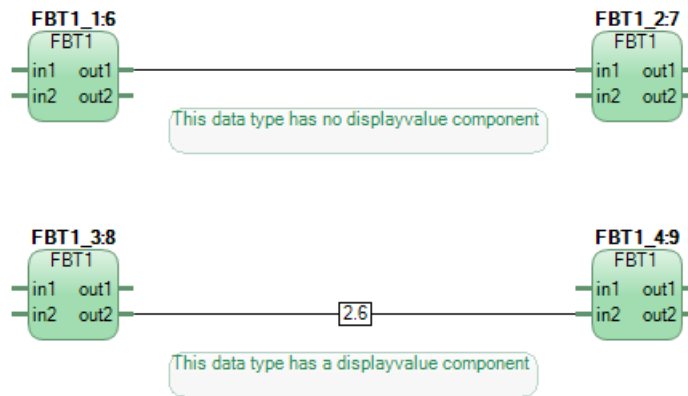


Figure 156. Graphical connections of structured data type

For example, in [Figure 156](#), both are graphical connections of structured data type but only the lower connection has a component marked as *displayvalue*.



The *displayvalue* attribute can be assigned to only one component in a structured data type.

It is also possible to assign this attribute through a combination of other attributes (for example, *coldretain displayvalue*, *nosort retain displayvalue*, and so on). In this case also, only one component can have this type of attribute (with *displayvalue*).

To view components in multilevel structures, the attribute *displayvalue* has to be set on every level, that is, the attribute has to be set not only for the simple data type to be viewed, but also for the sub-component the simple data type to be viewed belongs to, and so on.

To view or change the components of a structured graphical connection, right-click on the graphical connection and select **Show Online Values** from the context menu. The **Online Values** dialog box opens (see [Figure 157](#)).

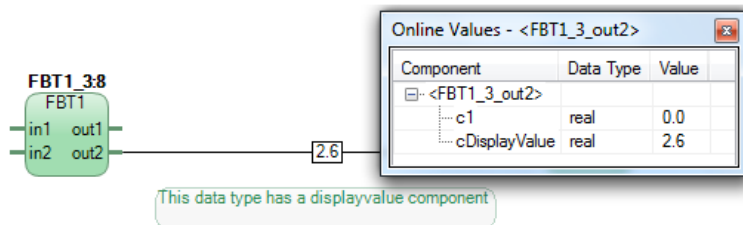


Figure 157. Online Values dialog box



It is possible to open several Online Value dialog boxes at the same time.



Tooltips are also available for online labels. This is useful if the online value is large so that it cannot be displayed as a label along with the graphical connection, or a truncated string value is displayed. Move the cursor over the graphical connection or the truncated online value label (if it is displayed) to view the tooltip that shows the whole value.

Dynamic Display of I/O Channels and Forcing

In test mode and online mode, you can use the hardware configuration editor for dynamic online display of I/O channel values and forcing.



The user must have Security Force I/O permission for the application, where the variable that is connected to the I/O is declared, to be able to force the I/O channel. For more information how to set permission, see the *System 800xA Administration and Security (3BSE037410*)*.

Forcing of I/O channels is performed in the hardware configuration editor under the *Status* tab, or in the POU editor in online mode. All I/O channels that can be connected to a variable in an application can also be forced in online mode, except for channels such as *UnitStatus* on each I/O unit and *AllUnitStatus* on the current

controller (see [Supervising Unit Status](#) on page 375).



If the application is SIL certified and runs in a High Integrity controller, I/O channels cannot be forced from Control Builder.

Normally, only channels with variable connections to application programs can be forced. However, if no variable is connected, you have to change the parameter *Copy unconnected channels* under the Settings tab for the current controller to obtain a status update. The I/O channels you can copy are *None*, *Inputs* or *Outputs*, or both the *Inputs* and *Outputs*.

When selected, the unconnected I/O channels are copied once a second so their status is available in the Status tab like normally connected I/O channels.



Copy unconnected channels is for test purposes only and should never be selected for a controller in a running plant, since it will increase CPU load.



To be able to force unconnected I/O channels, the user must have Security Force I/O permission for the hardware unit with the unconnected I/O channels. For more information how to set permission, see the *System 800xA Administration and Security (3BSE037410*)*.

Application programs requiring information about forcing and forced values, can use the I/O data types when connecting variables to I/O channels. In this way, you can use the *Forced* component (which indicates if the I/O channel is forced) and the *IOWalue* component (contains the value of the I/O channel) of the I/O data type.

When a channel is forced, all copying between the I/O value and the application value stops. The forced value is different for inputs and outputs. For inputs, forcing changes the variable value sent to the application. For outputs, forcing changes the physical I/O channel value. In this way, the application can see both the *Variable* (application) value and the *Channel* (I/O) value.

Forcing can be activated or deactivated using a check box in the *Forced* column for the channel. The background of the forced *Variable Value* changes to yellow to indicate forcing. To change the channel value, type in a new value for the *Variable Value*. This value overrides the values for the channel.

Channel	Channel Value	Forced	Variable Value	Variable
IXD.11.2.1	false	<input checked="" type="checkbox"/>	true	ShopDoors_ST.Normal.Photo_Cell
IXD.11.2.2		<input type="checkbox"/>		
IXD.11.2.3		<input type="checkbox"/>		
IXD.11.2.4				

Figure 158. I/O channel with the variable *Photo_Cell* forced to true.



More information is given in Control Builder the online help. Search the Index for “I/O”.

Forcing I/O Channels in SIL Applications

In SIL applications, forcing I/O channels are restricted. For each SIL application, you can define the maximum number of I/O channels that can be set in forced state. This setting decides the maximum number of connected I/O channels that can be forced at the same time.

To set maximum number of forced I/O channels

1. In Project Explorer, right-click the SIL application and select **Properties > Force**. A ‘Force Properties’ dialog open.

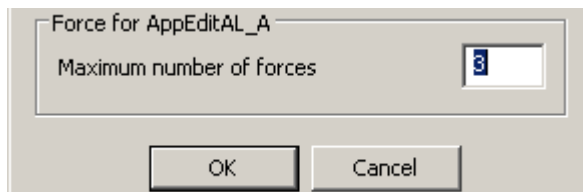


Figure 159. The ‘Force Properties’ dialog for setting the maximum number of forced I/O channels in a SIL application.

2. Set the maximum number of forced I/O channels and click **OK**.



The new value for the **Maximum number of forces** must **not** be less than the actual number of forced I/O channels in the running application. If the new value is less, a warning appears during download, and continuing the download leads to a controller shutdown.



Besides the Access Enable digital input signal, there are also two other signals that can be connected to the SM8XX module. One digital output signal for indicating that I/O channels are forced (normally a lamp) and one digital output signal for resetting all forced I/O channels (normally via a switch).



It is possible to release all forced I/O channels from the code by using the function block type `ForcedSignals` or the control module type `ForcedSignalsM`.

For more information on how to configure the `ForcedSignals(M)` types, see corresponding online help.



The `ForcedSignals` types are used to reset forced I/O signals in both SIL1-2 and SIL3 applications. Whereas, the `ForcedSignalsM` control module is used to reset forced I/O signals in SIL1-2 applications only. If a SIL3 application exists, the `ForcedSignals` function block should be executed from SIL3 application to reset both SIL1-2 and SIL3 forces.

Scaling Analog Signals

It is possible to temporarily change the scaling values for analog signals in online mode.



If scaling values for an analog signal are changed in online mode, the change will be lost if you enter offline mode, make configuration changes and then perform a download.



It is not recommended to change the scaling values `.Min`, `.Max`, or `.Inverted` of an analog signal from the code in a SIL3 application. Changing the scaling values might lead to a safety shutdown and a compiler warning is generated.

Supervising Unit Status

Each hardware unit has a *UnitStatus* channel that describes the current error status of the unit. Both dynamic and static warnings and errors are collected in this channel.

The data type, for the variable connected to the *UnitStatus* channel of the hardware unit, can be either of *dint* data type or of *HwStatus* data type. If a variable of *dint*

data type is connected to the *UnitStatus* channel, the possible unit status values are: 0 (OK), 1 (Error), or 2 (Warning).

The *HwStatus* data type contains the same information as shown under the *Unit Status* tab of the hardware configuration editor, that is, unit status information and status message acknowledgement functions. These components will be available by using the *HwStatus* data type as a variable connection to the *UnitStatus* channel.

In the example below, see [Figure 160](#), the *DO814UnitStatus* variable of *dint* data type is connected to *UnitStatus* of DO814 (unit status is 0=OK!). The *DI810UnitStatus* variable of *HwStatus* type is connected to *UnitStatus* of DI810 (*HwState* is 1, that is, unit status is Error).

Name	Current Value	Data Type	Attributes	Initial Value	I/O Address	I/O Description
DO814UnitStatus	0	dint	retain		Controller_1.0.11.1.19	Status of DO814
DI810UnitStatus		HwStatus	retain			
HwState	1	dint	retain			
HwStateChangeTime	2004-08-20-11:58:41.407	date_and_time	retain			
ErrorsAndWarnings	16#4	dword	retain			
ExtendedStatus	16#0	dword	retain			
LatchedErrorsAndWarnings	16#4	dword	retain			
LatchedExtendedStatus	16#0	dword	retain			

Figure 160. The UnitStatus connection gives access to the status of individual hardware units.



It is not possible to connect the *UnitStatus* channel on the hardware unit to a variable in a SIL1-2 or SIL3 application and obtain the status. However, the *UnitStatus* channel can be used in a HI controller configuration, but connect the channel only to a variable in a Non-SIL application.

Find Out What is Wrong by Using HWStatus

You cannot find out exactly what is wrong by using the simple data type *dint*, only that something is wrong. [Table 11](#) on page 125 shows that, in addition to using the *dint* type, you can also use the data type *HwStatus*. By using the structured data type *HwStatus*, instead of the simple data type *dint*, you may also find out what is wrong with the unit.

Among other things, the structured data type *HWStatus* contains the component *ErrorsAndWarnings*, which contains a bit pattern, representing the different errors that may occur in the unit. Each bit in the word represents a unique error.

Figure 161 illustrate how the component *ErrorsAndWarnings* in *HWStatus* can be accessed.

For example, the word takes the value of 16#80020000 (hexadecimal notation), if the CPU battery suffers from low voltage.



For more information on error codes, see Control Builder online help.

By combining *AC800MStatus.ErrorsAndWarnings* with the bit pattern 80020000¹ and using the *AND* operator, it is possible to trigger an error (or warning) from the hardware unit, together with the specific error code for “low CPU battery voltage”. The result is assigned to the boolean variable *BatteryLow*. The ST code for this condition is as follows:

```
(*Set the Boolean variable "BatteryLow" when AC 800M has low battery*)  
BatteryLow := (AC800MStatus.ErrorsAndWarnings AND  
16#80020000) <> 0;
```

In online mode it will be displayed as below in Figure 161.

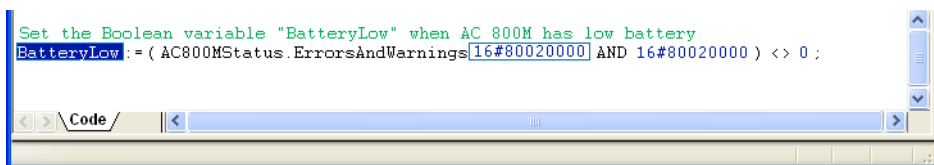


Figure 161. The variable *AC800MStatus* (of *HWStatus* type) has been used to access the component *ErrorsAndWarnings*.

AllUnitStatus

Each controller hardware object has one channel called *AllUnitStatus*, containing the summarized status of all hardware units added to the controller. The most serious unit status (dint) is forwarded up to the controller object, that is, the unit status of the controller is error if one unit has an error, and one has a warning.

1. Typed in ST editor in hexadecimal notation as 16#80020000.

AllUnitStatus can be used in the same way as *UnitStatus*, that is, the variable connected to *AllUnitStatus* can be of *dint* data type or of *HWStatus* data type.

Channel	Name	Type	Variable	I/O Description
IW0	AllUnitStatus	dint	ShopDoors_ST.Normal.HardwareStatus	Status of all hardware units

Settings Connections Unit Status

Row 1, Col 3

Figure 162. The *AllUnitStatus* connection gives access to the status of all units for a controller.

The variable connected to *AllUnitStatus* can be used in the application program, to write different conditions depending on status value (see *UnitStatus* Example Figure 161).

Binary Channels

Access All Inputs and All Outputs

Some units return a binary value, as a number of inputs divided on 8 or 16 channels. Typically, this applies to different types of sensors. These values can be collected via an overall channel, namely “All input”. This means that, instead of reading all variable values from each channel, one variable can be connected to the channel “All inputs” (IW0, see Table 11 on page 125), provided the variable is of *dword* data type. This technique can also be used for digital outputs. However, for digital output units, you must choose either to connect all individual channels or connect one variable to the channel “All outputs” (QW0, see Table 11 on page 125). You cannot use both methods simultaneously.



ISP and OSP values are not set for variables connected to *All Inputs/All Outputs*! ISP/OSP (Input/Output Set as Predetermined) will not work when using the channel “All Inputs” or “All Outputs”. I/O values will be lost in an error situation.

Check Channel Status

There are two ways to check the channel status for an I/O unit. You can either use the structured data type *BoolIO*, that is, read the component *Status* via *BoolIO*, or you can connect a variable of type *dword* to the “Channel status” (IW0, see Table 11 on page 125).

The component *Status* in *BoolIO* only gives you the status for that connected channel, whereas a variable of type *dword* that is connected to channel “Channel status” will read the status for all channels, given with bit 0 equivalent to channel 1, bit 1 equivalent to channel 2, etc. However, a variable of type *BoolIO* that is connected to each channel contains more information, since the component *Status* is a 32 bit *dword*, whereas *AllChannel* is a 16 bit *dword*. Connecting each channel to *BoolIO* gives more information, but also more variables to connect.

Connecting a variable to *AllChannel* will give you less information, but only one variable to connect.



Do not try to connect the component *Status* (inside *BoolIO*) directly to the channel. You must connect *BoolIO*. For information about connecting structured data types to IO channels, see [I/O Data Types](#) on page 124 and the variable example given in [Figure 37](#) on page 124.

Supervising Communication Variable Status

There are two methods to supervise the status of communication variables:

- Using the *:status* notation
- Using the *GetCVStatus* function

Supervising Communication Variable Using: Status Notation

For communication variables in non-SIL applications, the status can be supervised using the *:status* notation.

For example:

```
dword1:=CVMain:status;
```

In this example, the *:status* notation is used to obtain the status of the communication variable, *CVMain*. The status appears as *dword*.

See also, [Understanding the Complete Status Code](#) on page 383.



The *:status* notation shall not be used if the communication variable is in SIL1-2 or SIL3 application.

Supervising Communication Variable Using *GetCVStatus*

For communication variables in SIL1-2, SIL3 or non-SIL applications, the status can be supervised using the function, *GetCVStatus*, which is available in the System library. But for non-SIL, the *:status* method has better performance than the *GetCVStatus* function.

The *GetCVStatus* function accepts the communication variable as input, and provides the complete status and extracted statuses through different output parameters. These output parameters for extracted statuses can be connected to variables to control the logic (critical loop) in the SIL code.

The available parameters for extracting the status are:

- **Quality** – (dword) – The OPC quality of communication.
For GOOD, this parameter returns *16#C0*.
For BAD, this parameter returns *0*.
- **ServerSIL** – (dint) – The SIL of the server application (the application that holds the *out* communication variable).
If the server is SIL3, this parameter returns *3*.
If the server is SIL1-2, this parameter returns *2*.
If the controller that runs the server application is either hardware simulated or Soft Controller, this parameter returns the Expected SIL of the server, and not the actual SIL.
- **ManualAckRequired** – (bool) – Whether manual acknowledgment is required for the communication variable to restart the communication after failure.
This parameter returns *true* if manual acknowledgment is required, else it returns *false*.
- **ServerInOLU** – (bool) – Whether an online upgrade switch of the server is in progress, or whether the online upgrade switch is in progress on client controller (where the *in* variable is present).
This parameter returns *true* if an online upgrade of the server or client is in progress, else it returns *false*.
- **ServerIsSimulated** – (bool) – Whether the controller that runs the server application is either hardware simulated or Soft Controller.
This parameter returns *true* if the controller that runs the server application is either hardware simulated controller or Soft Controller. Otherwise, it returns *false*.
- **InternalStatus** (Internal Status Codes)
Status *WaitingToInit* (16#20) is indicated in the Quality during initialization. If the server is SIL2/SIL3 then external clients indicates *WaitingToInit* until three successful frames have been received. If the server is non-SIL, then external clients indicates *WaitingToInit* until the first successful frame has been

received. For internal clients `WaitingToInit` is indicated until the server has been downloaded. `WaitingToInit` is used for first download, cold download and after power-fail/short reset.

Understanding the Complete Status Code

The displayed complete status code of a communication variable depends on:

- The error (if any)
- The discrete values for the following factors:
 - Quality of communication
 - Server is simulated or not
 - Online upgrade in the server
 - SIL of the server
 - Manual acknowledgment is required or not, after error is generated

To understand the complete status code, first convert it from hexadecimal to binary (32 bits).

[Table 28](#) describes the bit-wise description of the used bits in the binary status of communication variable. Bit 0 is the least significant bit.

Table 28. Bit-wise description of communication variable status in binary code

Bit	Description
Bit 5	Not initialized. This bit is set in the following cases: <ul style="list-style-type: none"> • Initial download of application • Cold download of application • Restart of application after power-fail • Restart of application after short-reset This is reset when: <ul style="list-style-type: none"> • 3 frames are validated with success when ExpectedSIL is SIL2/SIL3 • First frame is received when ExpectedSIL is non-SIL • Out variable is downloaded for internal IAC
Bit 6 and Bit 7	Quality of OPC (<i>11</i> - GOOD, <i>00</i> - BAD)
Bit 8	The server is hardware simulated or it is a Soft Controller
Bit 9	Online upgrade switchover is happening in the server

Table 28. Bit-wise description of communication variable status in binary code

Bit	Description
Bit 10 and Bit 11	SIL of the server 11 represents SIL3 If Bit 10 is 0 and Bit 11 is 1, it represents SIL2)
Bit 12	Manual acknowledgment is required to restart communication after error generation.
Bits that represent communication errors⁽¹⁾	
Bit 16	The values are not communicated in time, but no timeout have occurred. The OPC quality bits are still good (11). This is used for external communication using IAC.
Bit 17	The values are not communicated in time, and a timeout has occurred. The OPC quality bits are bad (00). This is used for external communication using IAC.
Both Bit 16 and Bit 17 are set	The IP address has not been resolved for the communication variable.
Bit 18	The type does not match the type of the corresponding out variable. This is used for both internal and external communication using IAC.
Both Bit 16 and Bit 18 are set	The corresponding out-variable is declared, but not downloaded yet. This status is only relevant for internal IAC and occurs only when the out variable has existed and then has been removed.
Both Bit 17 and Bit 18 are set	The communication is not configured.
Bit 16, Bit 17, and Bit 18 are set	The task where the corresponding out variable is connected is not running. This is only relevant for internal IAC.
Bit 19	General error code
Bit 16 and Bit 19 are set	A fault in the safety frame verification was detected. This is only used in HI controllers.

Table 28. Bit-wise description of communication variable status in binary code

Bit	Description
Bits that represent protocol handling errors⁽¹⁾	
Bit 27	Wrong message type in the response message.
Bit 25 and Bit 27 are set	The out-variable cannot be found.
Bit 24, Bit 25, and Bit 27 are set	The Initiate request was unsuccessful.
Bit 26 and Bit 27 are set	The protocol version between client and server does not match.
Bit 24, Bit 26, and Bit 27 are set	The heap is full.
Bit 25, Bit 26, and Bit 27 are set	Permanent MMS error
Bits 24 to 31 are set	Unspecified error

(1) The status represented by a single bit is different from the status when this bit is set as a combination with other bits, in the same error category.

Status Indications



Status indications are not displayed in Test mode.

In the Project Explorer, dynamic status indications for the hardware units and tasks are displayed as shown below.

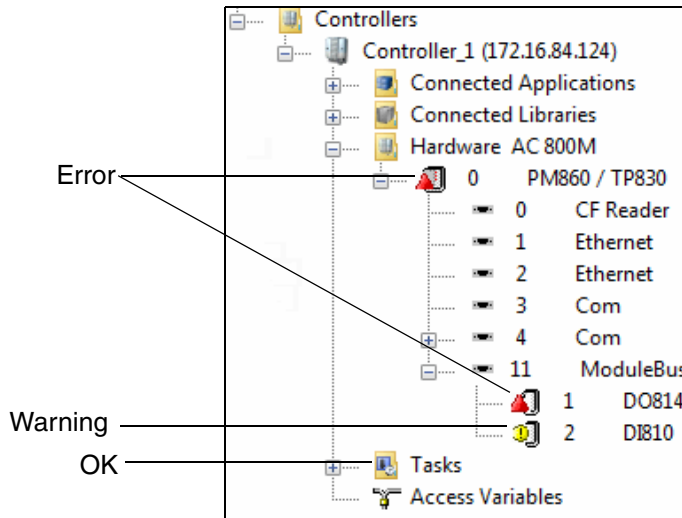


Figure 163. Status indications of hardware and tasks in Project Explorer.

- **OK**
No errors or warnings.
- **Error!** ▲
Hardware objects are marked with a red triangle icon if an error is detected in the hardware, for example, if a hardware unit is missing.
The task is marked with a red triangle when a serious error has occurred, for example, when a task is aborted as a consequence of too long execution time. The error is described in the Remark field of the Task Properties dialog. See [Task Abortion](#) on page 183 for more information.
- **Warning!** ⚠
Hardware objects are marked with a warning icon if there is an overflow or underflow at an analog channel, if the forcing of a channel is detected, or if an unacknowledged fault disappears. The task icon is marked with a warning icon

if the task is not used (“Not in use”), in the case of overload, or when the task is in debug mode and the task is halted, that is, non-cyclic mode (see Debug Mode in the *System 800xA Control AC 800M Getting Started (3BSE041880*)*). The warning is described in the Remark field of the Task Properties dialog. See [Task Control](#) on page 165 for more information about tasks.

An error has higher priority than a warning, for example, an error is indicated if an error occurs at the same time as channel forcing is detected.

A collapsed object folder shows status indications for all underlying objects, that is, status indication is always forwarded up to the controller icon. It is not until an object folder is fully expanded that you can be sure that status indications are shown next to the unit they actually belong to. If, for example, a single task has a warning, both its task folder icon and its controller icon are marked with a warning. Status indications are displayed up to the controller level only.

Acknowledge Errors and Warnings



Warnings concerning tasks do not have to be acknowledged.

All hardware unit errors and warnings have to be acknowledged by the user. Use the status tab in the hardware editor to obtain information about the error or the warning.



There is a possibility to acknowledge errors and warnings for all hardware subobjects by right click the main hardware object (Hardware AC 800M in [Figure 163](#)) and select **Clear Latched Unit Status**.

See Control Builder online help for more information about dynamic online display of I/O channel values and forcing and how to acknowledge errors and warnings.

Tasks



Changes to SIL applications are not allowed in online mode.

The *SetPriority* function does not work in a High Integrity controller.

Use the Task Overview dialog to display task information in online mode.

For each task, you can make changes to the Requested Interval Time, Offset, Priority and Latency using the Task Properties dialog. The maximum encountered intervals and the maximum encountered execution time can be reset.



It is not possible to change the task priority to/from 0 (Time-Critical priority) in online mode.

Debug mode can be used, but for debugging only. Functions based on the real-time clock (PID controllers, timers etc.) do not work properly when debug mode is used (also, see Debug Mode in the *System 800xA Control AC 800M Getting Started (3BSE041880*)*).



If debug mode is used in a running plant, task execution will be stopped.

You can also select **Always update output signals last in execution**, or select **Always update output signals first in next execution**.



Always update output signals first in next execution is not allowed for SIL3 tasks.



For further basic information about tasks, see [Task Control](#) on page 165. For Latency information, see [Latency Supervision](#) on page 181. See also Control Builder online help for how to carry out task changes.

Interaction Windows

An interaction window contains the graphics of a control module and is only accessible in online mode. An interaction window may contain both supervisory features, such as signal status, and interactive features, such as push buttons. The window can be accessed from:

- A control module in the Project Explorer.
- A function block in the Project Explorer. This is, however, only available under the condition that at least one control module exists and is connected to the selected function block type. By default, the first control module in the list will appear in the interaction window (this can be changed in offline-mode by right-clicking on the type name in the Project Explorer and selecting **Properties> Set Interaction Window Control Module**).

- An online editor containing a control module.
- An online editor containing a function block (compare with item 2 above).
- From interaction window objects in a control module.

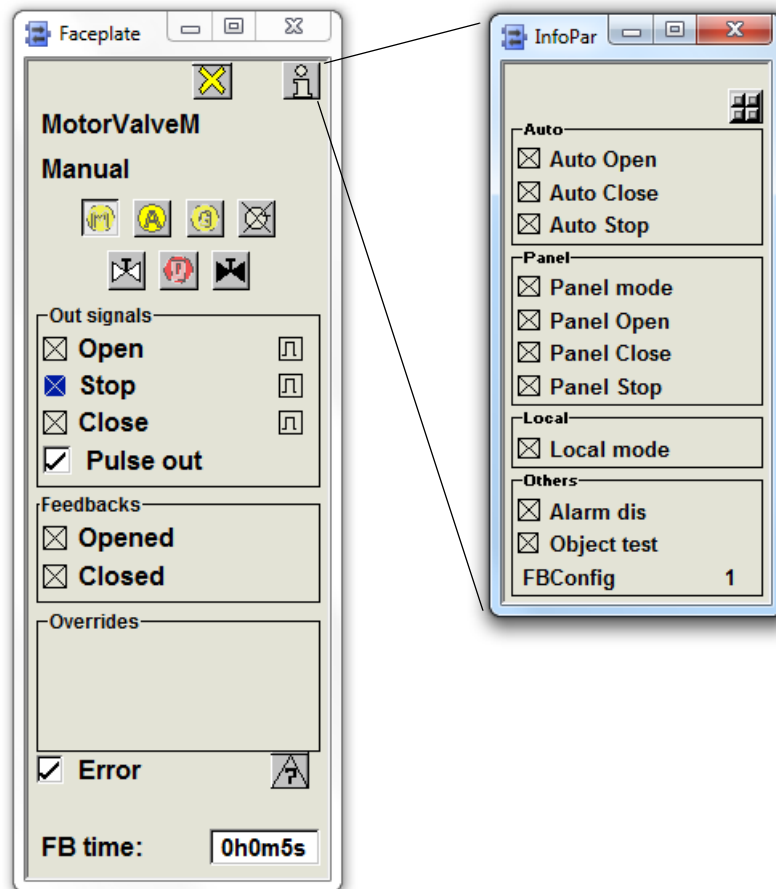


Figure 164. The left window is an interaction window activated from an application window interaction object. The right window (supervision only) appears after clicking the info interaction window button.

Status and Error Messages

There are function block types, control module types and functions that contain a parameter named *Status*. The *Status* parameter shows, in online mode and in test mode, a status code that correspond to a status message. The status code changes depending on the current state of the function block, control module or function.

There are function-specific status codes that are used within its range of application only, for example, communication-specific status codes. Some status codes are general and are used for most function blocks and control modules, and for functions with a *Status* parameter.



The different status messages are described in Control Builder online help.

Function block types and control module types with a *Status* parameter also have an *Error* parameter. The *Error* parameter is set to true if the *Status* parameter < 0, for example, if *Status* is -35 (Maximum size limit has been exceeded). Status codes >1 is used as warnings and do not set the *Error* parameter.

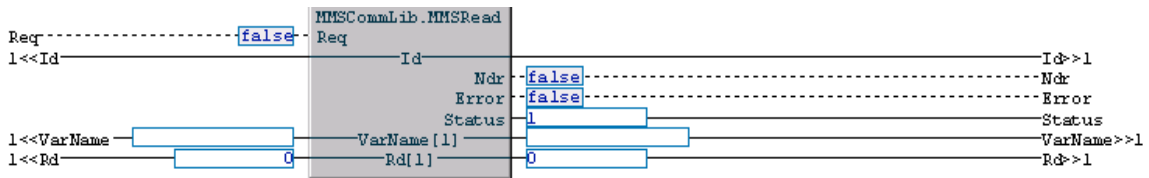


Figure 165. A function block with *Status* parameter and *Error* parameter (operation *successful=1*).

The *Error* and *Status* parameters can be used in the application program, for example, a condition can be written in the program for a specific status code.

Search and Navigation in Online and Test Mode

The Search and Navigation tool can be used to conduct simple searches and iterative searches when the project is in Online mode or Test mode.

This functionality makes it possible to search for input/output of a certain signal as a result of a single search, irrespective of name changes at parameter connections. This means all information concerning reading and writing from the whole Application/Controller(s) about a signal is found in the search.

The appearance of the Search and Navigation dialog in the Online mode and Test mode depends on the setting of the following options in the Search and Navigation Settings dialog box:

- **Allow editing of the Search Fields in Online/Test Mode**
- **Iterative searches in Online/Test Mode**

By default, the first option above is not set and the second option is set. See [Search and Navigation Settings](#) on page 208.

Iterative Search

In Online mode or Test mode, if the option **Iterative searches in Online/Test Mode** is set (the checkbox is checked) in the Search and Navigation Settings dialog, the iterative search hits are directly presented in one pane – the References pane.

See [Figure 166](#).

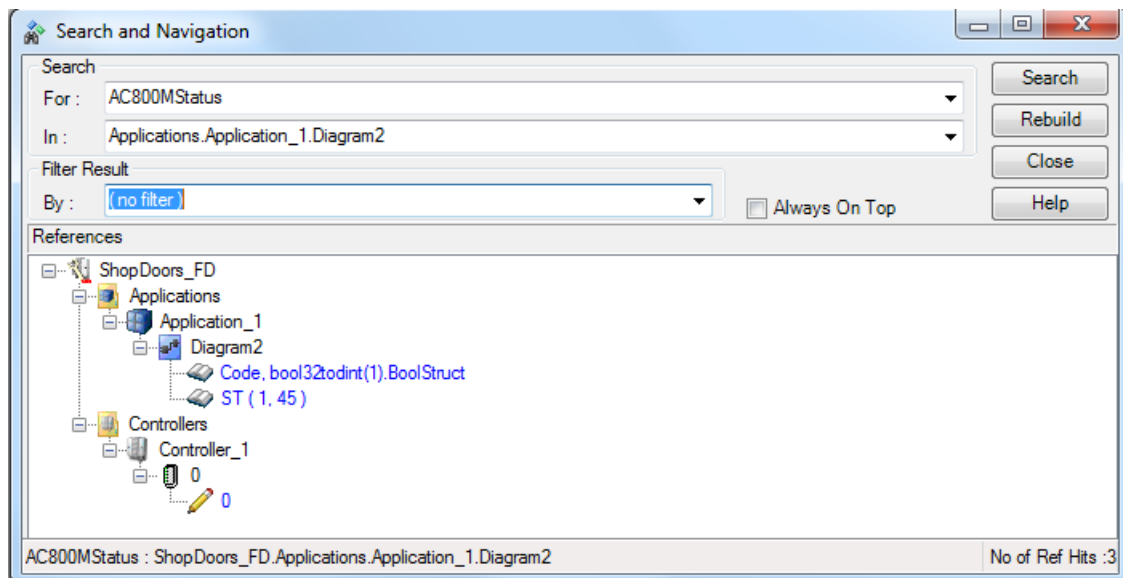


Figure 166. Iterative search results for the variable AC800MStatus in Online mode

In Online mode or Test mode, if the option **Iterative searches in Online/Test Mode** is not set (the checkbox is not checked) in the Search and Navigation Settings dialog, the search hits are presented in two panes—the Symbol and Definition pane, and the References pane. In this case, right click the symbol and select **Iterative Search** to start its iterative search. See Figure 167.

It is also possible to search for another item in the window and obtain the new results.

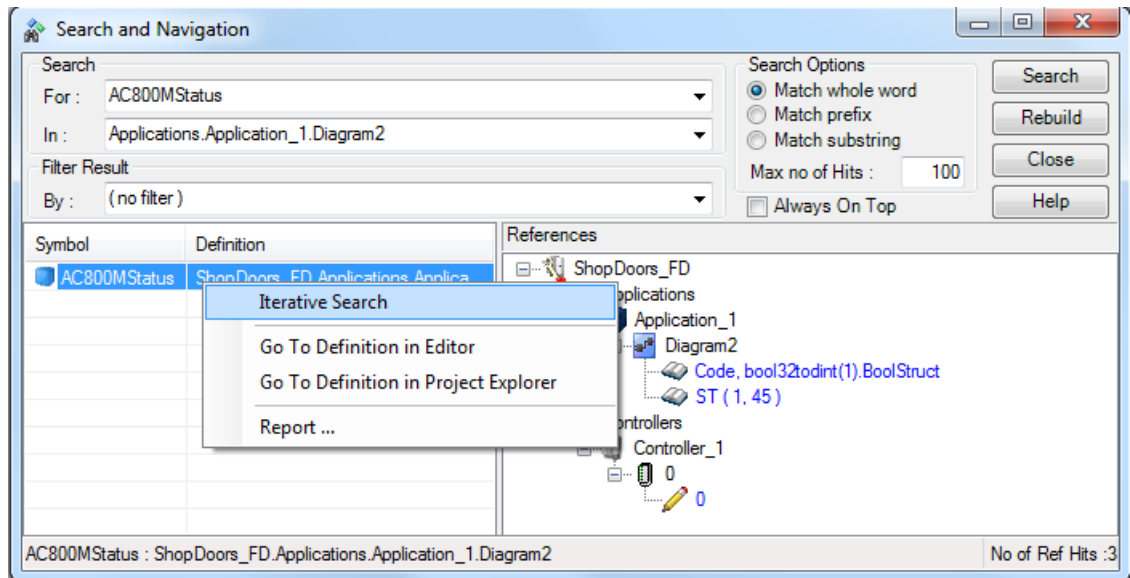


Figure 167. Search results for the variable *AC800MStatus* in Online mode, with the option for *Iterative Search*

After the **Iterative Search** option is selected (see Figure 167), the search results for the selected symbol are replaced by new search results in the References pane, which shows the header as *References (iterative search)*.

General Considerations for Search in Online/Test Mode

The tree view in the References pane shows where the signal is read or written.

It is possible to navigate from the Search and Navigation dialog to the references of a found symbol by double clicking a reference. Then a suitable editor is displayed and the symbol is highlighted in the editor.

The references are followed both upwards towards its first definition in a parent node, and downwards to the leaves of the project structure, in order to cover all usage. Every time a reference is followed, there is a new query to the search database. By means of those user hidden repetitive queries, all relevant information is collected from a single search.

There are following differences in online/test mode (compared to offline mode):

- Search In: drop-downs can only contain search paths for objects that you can see in online/test mode, for example, libraries cannot be searched.
- References only show information concerning where the symbol is used, as can be seen in online and test mode. The references tree (i.e. the tree presented in the references pane of the Search and Navigation dialog) presents instance paths in online mode and test mode.
- It is only possible to navigate to online editors and to the Project Constant dialog. The online editors that can be navigated to are the following:
 - POU editor
 - Connection editor
 - Control Module Diagram editor
 - Hardware configuration editor
 - Access variables editor
 - Diagram editor

In online mode, it is also possible to navigate from the Search and Navigation function to the corresponding object in the Project Explorer.

Project Documentation

Project Documentation in online mode is used to document (part of) the application tree in online or test mode. You can select any application object, set the “tree depth” in relation to the selected object, to document part of the tree only. You can also use filter conditions for a more specific search. Unlike the offline mode, the values of variables, parameters, etc. are included. For example, it is possible to filter out all coldretain variables and parameters in an application. The output is a Microsoft Word file, hence Microsoft Office must be installed.



All project documentation will be connected to a standard template.

1. Enter online or test mode and select an application object in Project Explorer.
2. Select **File > Documentation Online...** to open the Project Documentation dialog.

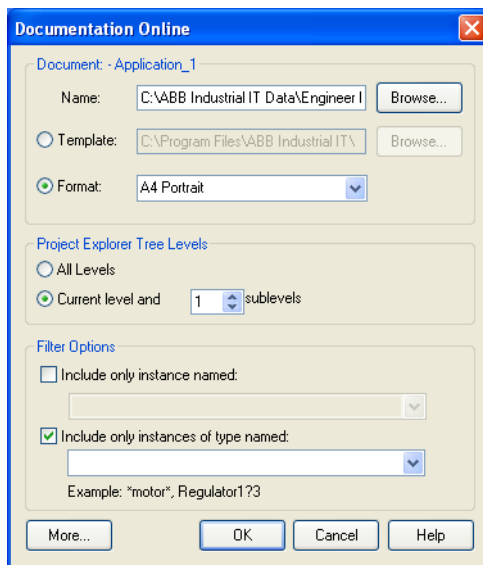


Figure 168. The Documentation Online dialog.

3. See Control Builder online help for information about dialog settings and selections.



See [Project Documentation](#) on page 253 for information about Project Documentation in offline mode.

Section 5 Maintenance and Trouble-Shooting

This section provides important information for maintenance and trouble-shooting Control Builder products. It mainly advises you on how to maintain your system, and how to collect information from a malfunctioning control system. The latter information is particularly valuable if your supplier's service department is to be involved.

Introduction

Software maintenance and trouble-shooting includes the following activities:

- Remote Desktop Connection [Running Control Builder on Terminal Server](#) on page 398 describes how to start and run the Control Builder Professional as a terminal session on a Terminal Server.
- [Backup and Restore](#) on page 401 gives a short overview of backup and restore of an 800xA system. For detailed instructions on how to upgrade or restore a complete system, see 800xA system documentation.
- [Migration](#) on page 409 describes how to migrate from Compact Control Builder to 800xA and how to migrate from 800xA to Compact Control Builder, within same system version.
- [Import and Export](#) on page 413 describes how to import and export libraries, programs and individual objects.
- [About Library Import/Export](#) on page 418 points out a number of things that are of importance from a maintenance perspective.
- [Controller Configuration](#) on page 422 describes how to configure handling and logging of system alarms and events, using the Error Handler.
- [Error case handling when I/O Channels have returned to OSP](#) on page 446 lists the requirements, preparations and the steps of the process.

- [Trouble-Shooting](#) on page 446 lists a number of error symptoms, and suggest actions upon these.
- [Error Reports](#) on page 493 describes how to write a complete error report, so that the support engineers get a complete picture of an error situation.

Running Control Builder on Terminal Server

It is possible to connect to a terminal server and run the Control Builder Professional as a terminal session, by using Windows standard Remote Desktop Connection. All you need is network access and permissions to connect to the other computer.

There are some restrictions when using Control Builder Professional as a terminal session:

- A maximum of 10 concurrent Control Builder sessions can run on the terminal server.
- There can be only one active Control Builder session per interactive Windows user.
- It is not recommended to run a Soft Controller on the terminal server.

Characteristics of Control Builder as Terminal Server

There are some things that differ a Control Builder terminal session from a locally executing Control Builder session.

MMS Process Number

The MMS process number of the Control Builder process is usually 1. It will still be 1 for a Control Builder session that is executing locally on the terminal server console. For a remote Control Builder session the MMS process number will be in the interval of 31-40. The MMS process number will be 31 for the first remote Control Builder session, 32 for the second and so on.

The MMS process number can be shown, for example, by selecting **Help > About Control Builder M Professional** in the Control Builder.

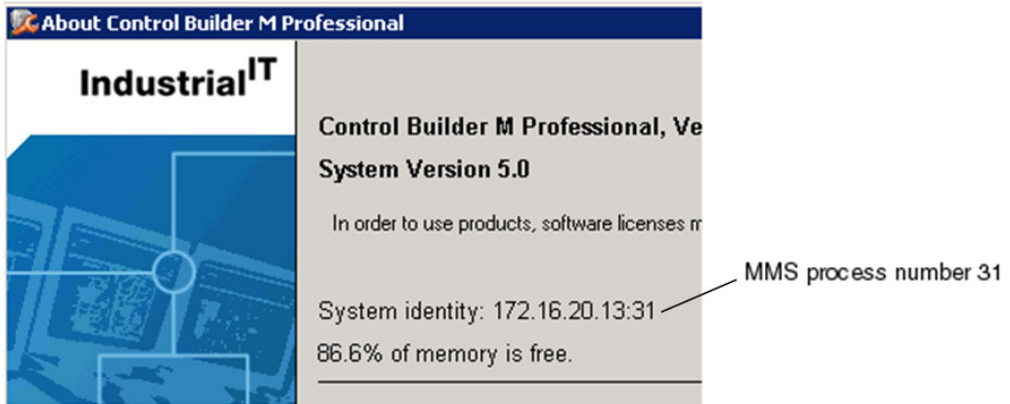


Figure 169. About Control Builder M Professional dialog

Working Folder

In a standard installation of Control Builder the working folder is *C:\ABB Industrial IT Data\Engineer IT Data\Control Builder M Professional*. For each remote Control Builder session *...\Terminal Sessions\”UserId”* will be added to that path. “UserId” is the user id of the interactive Windows user logged on to the terminal server. This is only valid for remote sessions. A Control Builder running on the terminal server console will use the normal working folder.

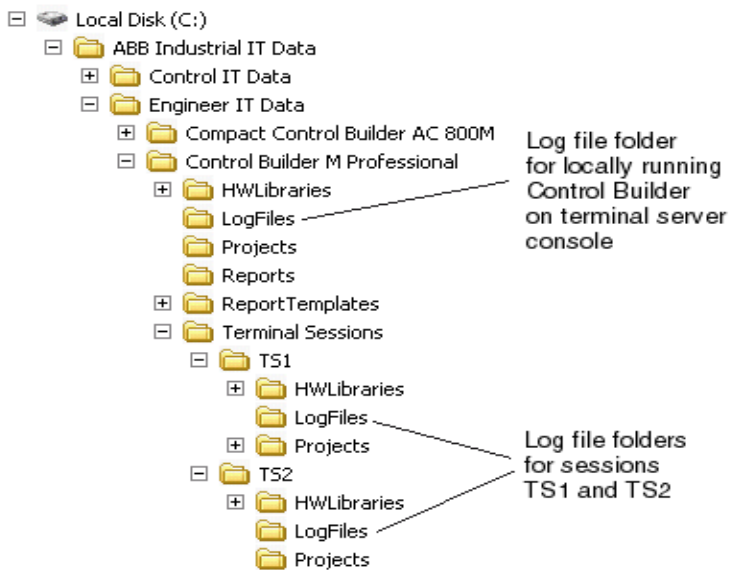


Figure 170. TS1 and TS2 users added as Control Builder terminal sessions.



The working folder can be changed by using the Control Builder Setup Wizard. A remote Control Builder session will then add *...\Terminal Sessions\”UserId”* to the specified file path of Working Folder for File Locations, in the Setup Wizard.

Note that each Control Builder terminal session has a LogFiles folder, where the log files are saved for that particular session. Log files for a Control Builder running locally on the terminal server will be found in the LogFiles folder under the Control Builder M Professional folder.

Remote Session Indication

When dealing with support engineers it is important to know if the Control Builder is running as a remote session or as a local session. A small visual indication together with the user name will be shown in the status bar of the Control Builder, if it is running as a remote session.

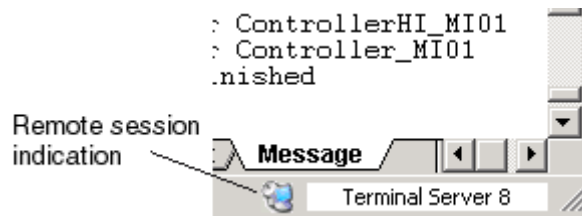


Figure 171. Remote session indication in Control Builder

Backup and Restore

Introduction

The Backup function prevents data loss if a complete system crash should occur. The Backup function saves the complete system on a local disk.



For instructions on how to perform a backup or restore, see the *System 800xA Tools (2PAA101888*)*. Coldretain, structure and domain files are included at backup/restore.

For backup of individual projects, applications, libraries, etc., the Import/Export function should be used, since this function handles individual files. The Import/Export function is described in [Import and Export](#) on page 413.

Files for Separate Backup

There are some settings files that are stored locally. These need to be backed up separately:

- **OPC Server Configuration and System Setup Files**
The OPC Server stores configuration files (*.cfg) and system setup files (*.sys) on local disc. These files are stored in the OPC server working directory and need to be manually copied to safe media on a regular basis. See the *AC 800M OPC Server (3BSE035983*)* for more information.
- **Control Builder Settings File**
Each Control Builder client saves its settings in the file `systemsetup.sys`. This file is saved on local disk, in the Control Builder working directory, and has to be manually backed up to safe media on a regular basis.

Remove and Add FSD Server Files

Could retain files, and files associated with Control aspects, such as applications, controllers and projects, are stored in the File Set Distribution (FSD) server in 800xA. The *FSD* tool makes it possible to view, add, extract and delete files that are stored in the FSD server.



This tool should be used with extreme caution, since a mistake when deleting or changing files in the FSD server might cause serious problems.

To start the tool, go to the Windows Start-menu and select **ABB Start Menu > Engineering > Utilities > FSD Tool**.

Some examples of how to use the tool:

- Extract files and store them on local disk for further examination.
- Replace lost or corrupted files.



For more detailed information on the tool, see its online help, which is opened from inside the tool.

Compiler Output File Helper

The Compiler Output File Helper tool is able to restore compiler output files from a folder on a disc to an 800xA Aspect Server. All files belonging to a control builder

project, or all files belonging to one or many controllers of a project, can be restored in one single operation. This function is valuable in some upgrade use cases as well as when the simulate hardware function is used.



The FSD Tool is also able to extract and restore compiler output files. The difference is that Compiler Output File Helper tool can extract all needed files in one operation, which is more convenient to use.

One situation when the tool is valuable is when a system will be upgraded to a new system version. Then the production system is backed up and restored on another PC using the new system version. The upgrade process might also be performed in another location, that is an office, and can take some time to accomplish. The original production system is kept running and connected to the controllers until the upgrade process is finished. If the maintenance personal is forced to change and download an Application or Controllers Configuration, then the compiler output files is stored in the original production system. Here the tool can be used in order to extract the compiler output files, of correct version, on a disc. Later on, the tool can be used in order to restore the files into the new upgraded system.

The CompilerOutputFileHelper tool can be opened from Engineering & Development\Control Builder M\Tools\CompilerOutputFileHelper in installation media. The tool should run on any engineering client. There are two different exe files: CompFileHelper20.exe and CompFileHelper45.exe. The CompFileHelper20.exe requires .NET Framework 2.0 and is targeting SV5.0 SP2 up to all variants of SV5.1 provided that the appropriate .Net framework is installed on the PC. The CompFileHelper45.exe requires .NET Framework 4.5 and is targeting SV6.0 and future versions. The exe file are dependent on some help DLLs according to [Figure 172](#).

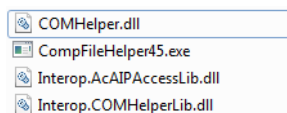


Figure 172. DLLs for CompilerOutputFileHelper

The Compiler Output File Helper tool extracts compiler output files from an 800xA Aspect Server and stores the files in a folder on a disc. All files belonging to a

control builder project, or all files belonging to one or many controllers of a control builder project, can be extracted in one single operation.

Extract Compiler Output Files for Project

1. Select the System and Control Builder Project from the drop down boxes as shown in the [Figure 173](#).

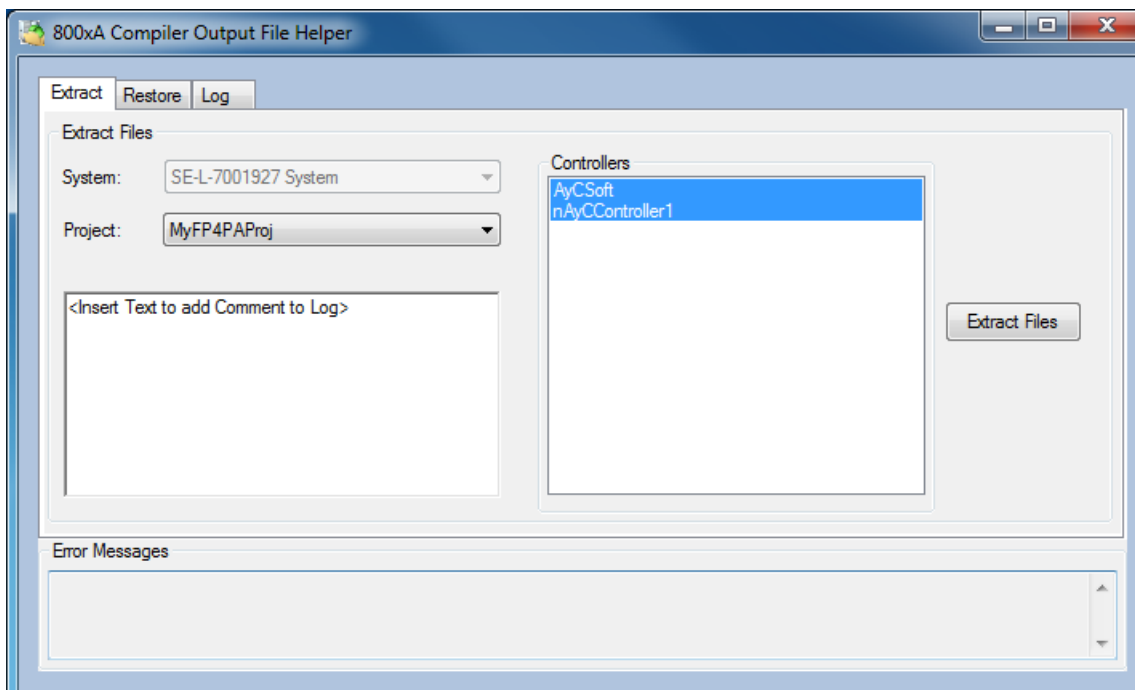


Figure 173. Extract Tab of the Compiler Output File Helper

2. The controllers of the selected project are shown in a list box. One or many controllers can be selected in the list box. An edit box provides the possibility to add text to the log file.

3. Click the **Extract Files** and a file browser dialog appears as in [Figure 174](#).

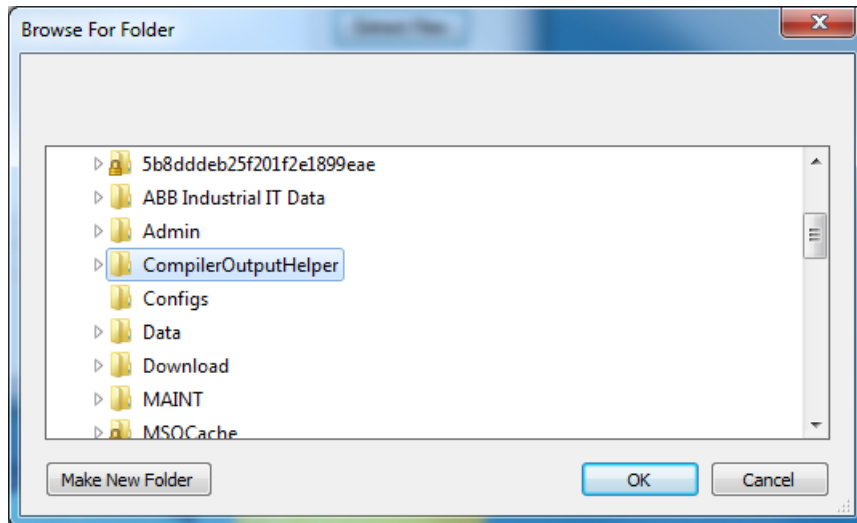


Figure 174. File Browser Dialog

4. Select a **CompilerOutputHelper** folder and click the **OK**. The tool extracts all compiler output files belonging to the selected controllers and the applications associated with the controllers. The tool creates a new sub directory each time the **Extract files** is activated.

Restore Files using Compiler Output Helper

1. In the Restore tab click **Browse** to open the file browser dialog as shown in [Figure 174](#).
2. Select an earlier created folder from the file browser. Information about path, computer name, project and date are shown as in the [Figure 175](#). In addition, a

list box with controllers is displayed. One or many controllers can be selected in the list box. An edit box provides the possibility to add text to the log file.

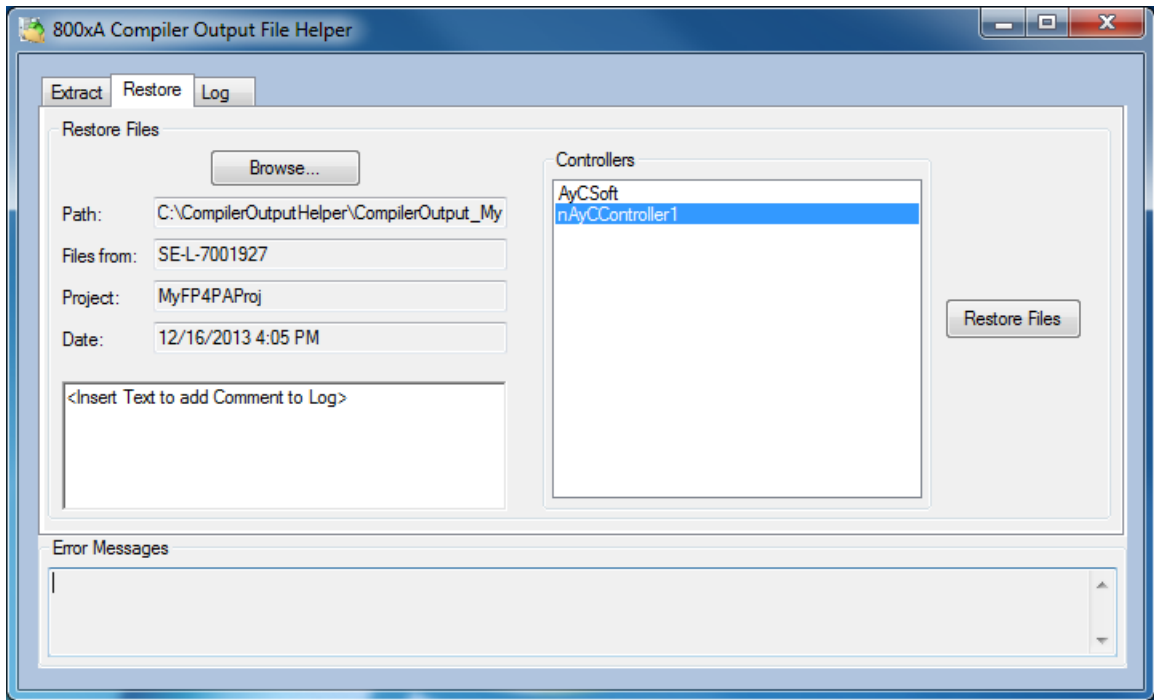


Figure 175. Restore Tab of the Compiler Output File Helper

3. Click **Restore Files** and a Confirm action dialog as in [Figure 176](#) is shown. Click **Ok** in the confirmation dialog in order to restore the files.

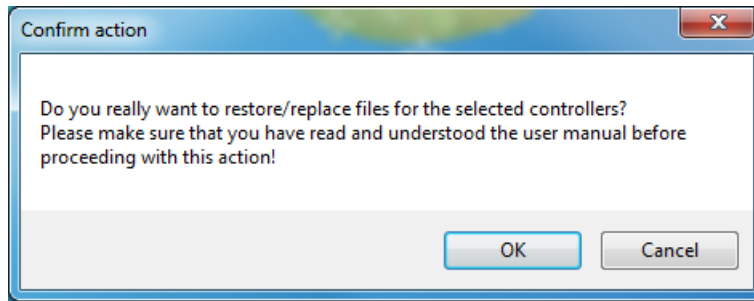


Figure 176. Confirm action dialog

Log Tab in the Compiler Output File Helper

This Log tab displays the current log as in the [Figure 177](#). The SessionLog.txt file is also available in the CompilerOutput folder.

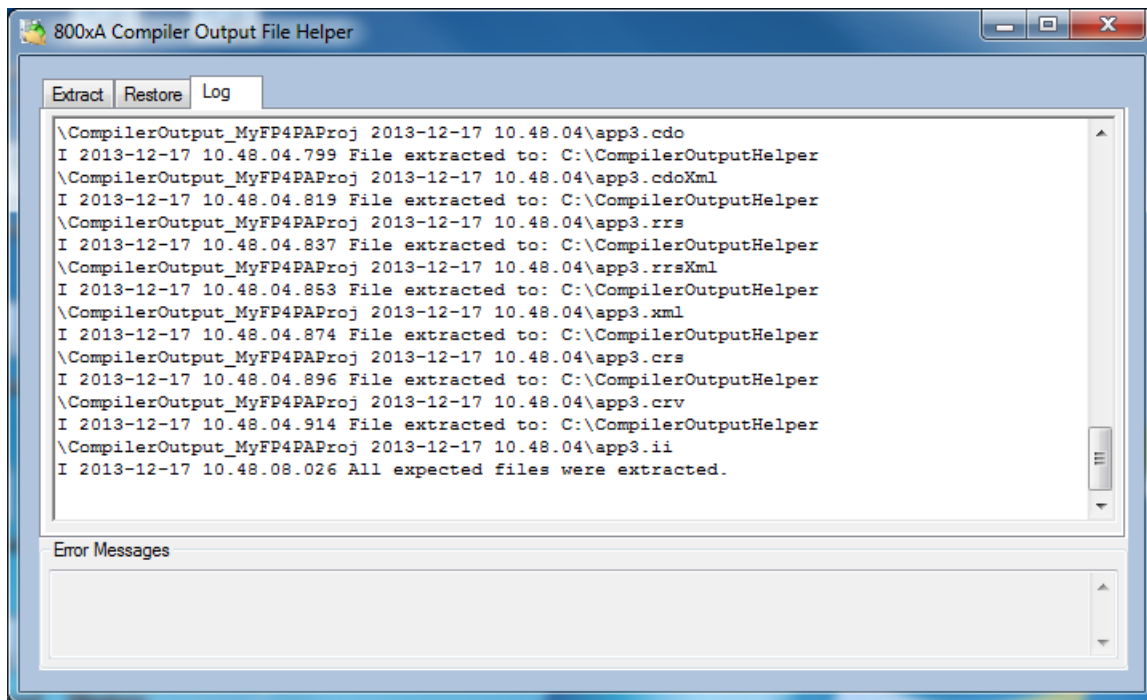


Figure 177. Log Tab of the Compiler Output File Helper

Migration

Within the same system version, it is possible to migrate a project from 800xA to a Compact Control Builder project as well as to migrate a project from a Compact Control Builder to a project in 800xA.



Only complete projects can be migrated. It is possible to migrate smaller objects, such as single libraries, applications, controllers or types, by packaging them within small or empty projects.

Migration from 800xA to Compact Control Builder

When migrating a project from 800xA to Compact Control Builder there are some things to consider:

- It is not possible to migrate projects containing High Integrity controllers or controllers containing a PM865 CPU.
- It is not allowed to migrate a project containing more than one version of a user-defined library. However, it is possible to migrate projects containing several versions of standard libraries.
- Projects containing controllers with undefined hardware units cannot be migrated. The migration will be aborted and the user has to define all undefined hardware units before a migration can be done.
- A project containing hardware that is not supported in Compact Control Builder, for example CI860, will be migrated. When the migrated project is opened in Compact Control Builder error messages are displayed and all hardware types of not supported hardware units will be displayed as undefined hardware units in Compact Control Builder.

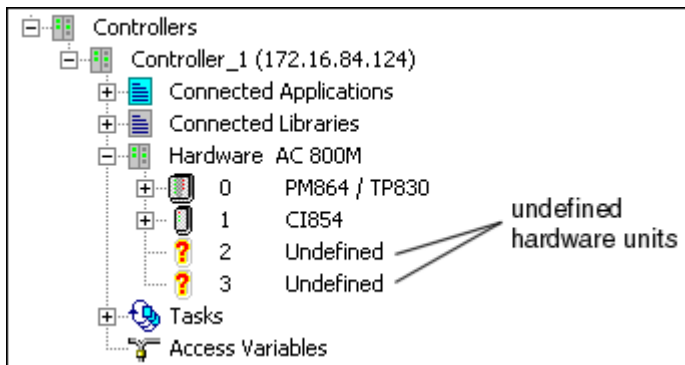


Figure 178. Undefined hardware units in Project Explorer.

- Online files, such as Cold Retain files and Difference Report files, are not migrated.

To migrate a project in 800xA to Compact Control Builder, open the project to be migrated in Control Builder Professional and select:

Tools>Maintenance>Compact CB>Save in Compact CB Format.

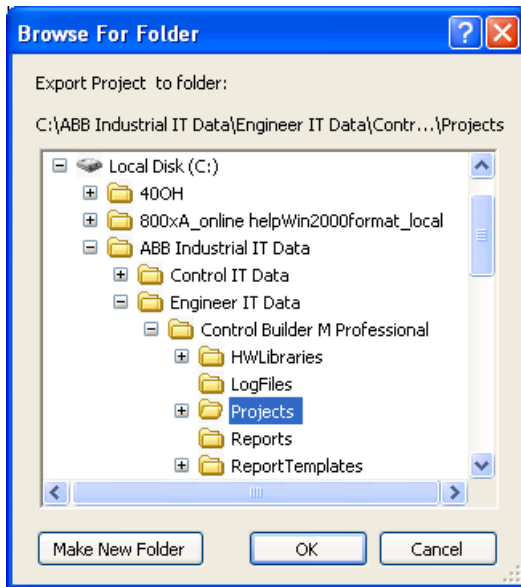


Figure 179. Save in Compact CB Format dialog

A dialog to select where to save the project is opened. The migrated project will be placed in the selected folder, together with all applications and controllers within the project. All user-defined libraries (both POU and hardware) will also be placed in this folder.



To make the OPC server work properly, it is recommended to place the folder with the migrated project in the configured project folder of Compact Control Builder, before the migrated project is opened.

Migration from Compact Control Builder to 800xA

When migrating a project from Compact Control Builder to 800xA there are some things to consider:

- Before migrating, make sure that all used libraries in the project are available. Move/copy the complete installed project structure, including the Libraries folder, to the location where you want to do the migration from.

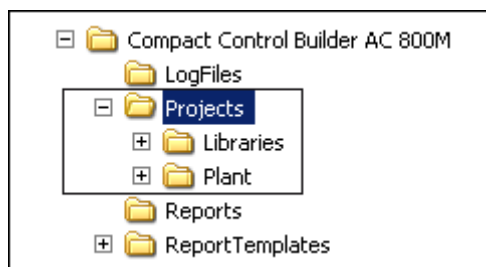


Figure 180. Project structure with “Libraries” and the “Plant” project.

- It is only possible to migrate a project once.
- It is not possible to use the migration to overwrite anything that already exists in 800xA.
- Projects containing controllers with undefined hardware units cannot be migrated. The migration will be aborted and the user has to define all undefined hardware units before a migration can be done again.
- A user-defined library is opened from Compact Control Builder format and written to 800xA, if it does not exist in 800xA. If a user-defined library exists in 800xA it is always read from 800xA, no matter what state it has.

- The migration is aborted, if the project to be migrated has a library that contains different types and it has same name as a library that already exists in 800xA. The library has to be renamed before it can be migrated again.
- If a standard library is missing in 800xA, the migration is stopped. The missing library has to be installed in 800xA before a complete migration can be done.

To migrate a Compact Controller project to 800xA, start Control Builder Professional and select:

Tools>Maintenance>Compact CB>Open from Compact CB Format.

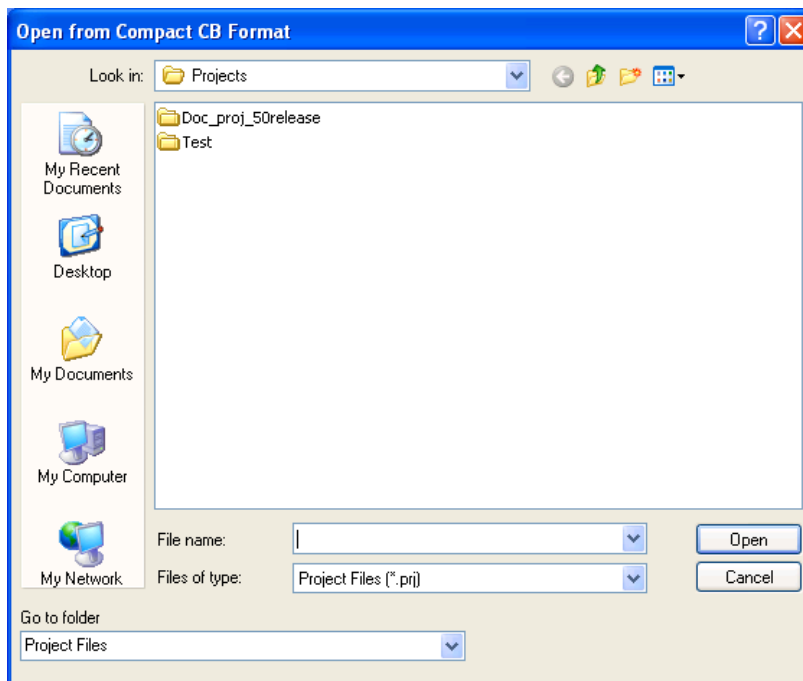


Figure 181. Open from Compact CB Format dialog.

A dialog to select the project (*.prj) to be migrated is displayed.

If more than one system and/or more than one environment are available, a dialog where to select destination system and destination environment is displayed, after the *Open from Compact CB Format* dialog.

Import and Export

Introduction

The import and export function is used for selective backup of entities such as projects, applications, libraries, etc. or for moving solutions between systems.

For backup of a complete system, use the Backup/Restore function. The Backup/Restore function is described in [Backup and Restore](#) on page 401.

Any entity of your automation solution can be exported for import at a later stage. The export is stored as an afw file.

Exporting an entity can be done either with or without dependencies. If dependencies is included, the export file will be consistent and include everything needed to be able to import it at a later stage.

Exporting an entity can be done with or without children. For normal maintenance, the selection with children shall always be used.

[Import and Export Alternatives](#) on page 416 shows how to export and import entities and the differences between exporting with or without dependencies/children.



See also [Applying Cold Retain Values when Importing Applications](#) on page 417.

Export a Library



The user defined permissions that are used to configure property permissions are not included when exporting the entities (project, application, library etc.) with or without dependencies.

The user defined permissions have to be manually exported separately by the user.

This option lets you backup/export a single library. When you are done, an afw file is created. This file can be used to import the library into other systems.



For information on how to change the development state of a library, see [Library Management](#) on page 135.



When importing a library that already is present in the system, types that are not existing in the afw file will be removed automatically. Use the **Show Differences** alternative on the **Import/Export** menu to find these types, then delete them manually.

If a library is already present and its status is Closed or Released, it cannot be imported.

Make sure you are in the Library Structure:

1. In the Library Structure, select the library version (for example MyDevLib 1.0-0). The aspect pane opens.
2. Select the **Library Version Definition** aspect and click the **General** tab. The aspect preview pane opens.

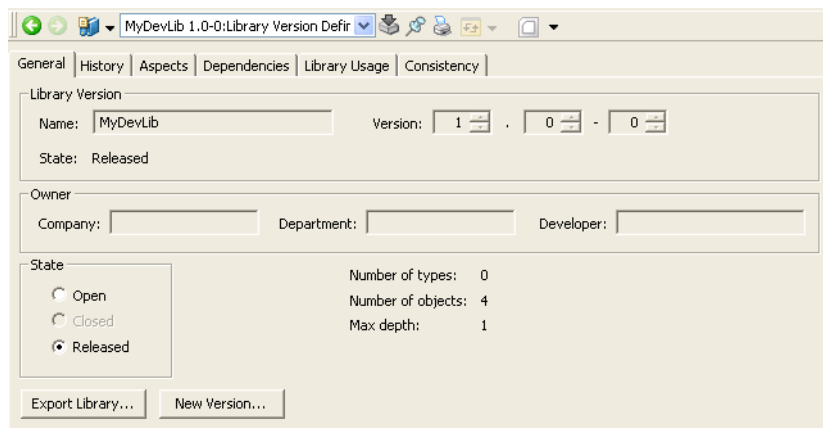


Figure 182. The Library Version Definition aspect preview pane, with the State selected as 'Released'.

3. Click **Export Library**. The Export Library dialog opens.
4. Name the library file (it is advisable to use the library name, for example MyDevLib) and click **Save**. Plant Explorer saves an afw library file.



Plant Explorer will display the text *Library export succeeded* under the Library Archive button, when done. The library will be saved as an afw file (in this example MyDevLib.afw)

Export an Application/Controller

You can also Backup/Export an application or a controller. Drag the application/controller object and drop it onto the Import/Export window, then select whether to export with dependencies and/or children.

Import an Application/Controller

Rollback Application/Controller Version

To rollback to an application/controller backup, import the backup MyApplication/MyController.afw file.

Re-import of Project



The Applications/Controllers that are added in the project after exporting the project, are not deleted if the exported project is imported again. If required, they can be deleted manually.

For more information on how to use the Import/Export Tool, see the *System 800xA Tools (2PAA101888*)* manual.

Import and Export Alternatives

To reach an expected result when exporting/importing, the following table can be used as a guide.

Table 29. Export settings

Object	Export Settings	Result (afw file)	Notes for Import
Project	Including dependencies and children.	Entire Project including applications, controllers and user-defined libraries.	Answer “No” to import the Control Network object.
Project	No dependencies, including children.	Entire Project including applications and controllers.	All libraries used in the project must exist in the system when importing
Application	No dependencies, including children.	Entire Application without connected libraries.	Connected libraries must exist in the system when importing.
Application	Including dependencies and children.	Entire Application, Project object plus all user-defined libraries inserted in the project.	Choose to not import the Project object and Control Network object.
Controller	No dependencies, including children.	Entire Controller (without connected libraries) plus connected application objects (only the positions of the applications, not the entire application entities).	Connected libraries and applications must exist in the system when importing.
Controller	Including dependencies and children.	Entire Controller, all connected applications, the Project object, and all user-defined libraries inserted in the project.	Choose if you want to import the Project, connected applications and libraries.
Library version	Including dependencies and children.	Entire Library plus all user-defined libraries, connected to the library.	

Table 29. Export settings (Continued)

Object	Export Settings	Result (afw file)	Notes for Import
Library version	No dependencies, including children.	Entire Library without connected libraries.	Connected libraries must exist in the system when importing.
Library version	Export Library button in Library Version Definition aspect.	Entire Library without connected libraries.	Connected libraries must exist in the system when importing.
Control Module Type / Function Block Type / Diagram Type	No dependencies, including children	Only the type.	If the type has any formal instances, the formal instances types must exist in the system when importing.
Control Module Type / Function Block Type / Diagram Type	Including dependencies and children.	Entire application or library where the type is placed, including the application or libraries dependencies.	



By default, the VMT application under a High Integrity controller is not included in the export, when the High Integrity controller is exported. If this controller (from the Production system) is to be re-imported, the VMT Application must be manually included in the export.

It is recommended to always include all the applications that are connected to the controller while exporting.



Entities in other structures (for example, the Functional Structure) are not included in the export even if it is done including dependencies. AC 800M entities existing in Functional Structure (for example) must then be exported separately.

Applying Cold Retain Values when Importing Applications

When importing an application the cold retain files are imported to the system only if the application does not exist in the system or if the application exists but no download has been made (no cold retain values exist in the system).

This applies when importing to a non version handled system (i.e. a system not having Configure-Deploy Support enabled) or when importing to the production environment of a version handled system.

The cold retain values are never imported when importing an application to the engineering environment of a version handled system. It is possible to force an import of the cold retain values to the system (except to engineering environment). To do that the cold retain files (.crs and .crv files) associated with the application should be removed using the FSD Tool. When no changes (apart from saving new cold retain values) have been made in the application since when the export was done a dummy change of the application in the system is necessary in addition to removing the associated files.

About Library Import/Export

Library management is described in [Library Management](#) on page 135. However, there are some things that are worth emphasizing from a maintenance perspective:

- When importing and exporting libraries, it is of importance which version of a library that is used. If a library is imported that depends on another version than the one already in the system, one of two things will happen:
 - If the library was exported with all libraries it depends on, then you will simply get two versions of the same library in your system.
 - If the library was exported without dependencies, there will be an error and your imported library will not work.
- If a library is exported without having reached the development state Released, there is a risk that there might be two libraries with different content, but with the same version number. If the “wrong” library is imported, then serious problems might arise.



If you need more information about libraries and library maintenance, refer to *System 800xA Control AC 800M Binary and Analog Handling (3BSE035981*)*.

Detailed Difference Report During Import

The detailed difference report shows data differences from the aspects.

A number of aspect types support detailed difference report, enabling the comparison of data differences during import.

The aspect types that are supported include Control Module Type, Function Block Type, Diagram Type, Single Control Module, Data Type, Application, Program, Diagram, Project, Controller, Library, Hardware library, Hardware Type, Hardware unit, Task, Access variables, and Project constants.

For more details about Detailed Difference Report refer to *System 800xA Maintenance (3BSE046784*)* manual.

Start Values Analyser

The StartValuesAnalyser collects the runtime values and prints the path, init value, communicated value, quality, and type for all cold retain and retain variables /parameters of a Control Builder application running in the controller(s). The Start Values Analyser is also able to compare runtime values from different occasions and to print out the differences. If a tunable variable/parameter is configured as retain, instead of cold retain, then the tuned value will be lost after a firmware upgrade of a controller. The Start Values Analyser is able to propose a list of cold retain candidates. This function is available for version 6.0 and forward.

The Start Values Analyser tool can be opened from Engineering & Development\Control Builder M\Tools\StartValuesAnalyzer in installation media. There are two different exe files: StartValueAnalyzer20.exe and StartValueAnalyzer45.exe. The StartValueAnalyzer20.exe requires .NET Framework 2.0 and is targeting SV5.0 SP1 up to all variants of version 5.1 provided that the appropriate .Net framework is installed on the PC. The StartValueAnalyzer45.exe requires .NET Framework 4.5 and is targeting version 6.0 and future versions. The tool makes use of an AC 800M OPC Server. So it is appropriate to run the Start Values Analyser executable on a PC with an OPC Server.

Data Collection from Controllers

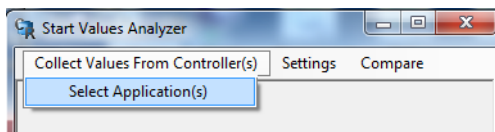


Figure 183. Select Application(s)

Click **Collect Values From Controller(s)** and select **Select Application(s)**. Then the **Select Application(s)** dialog opens as in the [Figure 184](#).

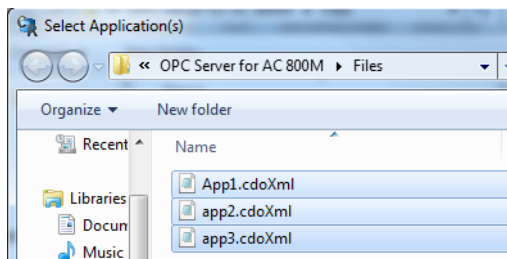


Figure 184. Select Application(s) Dialog

The tool automatically browses to the default files folder of the OPC Server. The application compiler output files needed by the OPC Server are present in this folder. It will then analyze the selected cdoXml file and the corresponding rrsXml file of each selected application. These files contain information about all POU's, variables, parameters, cold retain properties, initial values and so on, for the current version of an application. It connects to the OPC Server and communicates the runtime values of the cold retain and retain variables/parameters and prints out the path, initial value, communicated runtime value, quality and types the variables/parameters to files.

The tool creates a new directory each time the Select Application(s) dialog is executed.

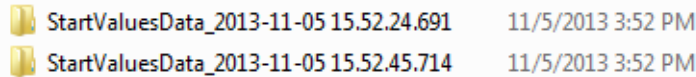


Figure 185. StartValuesData Folder

For each selected application the respective output files are created as in the [Figure 186](#).

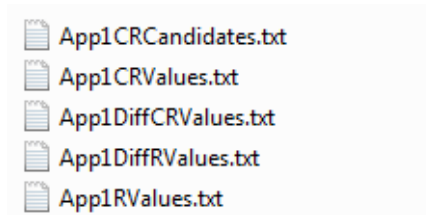


Figure 186. StartValues Output Files

- The CRValues file contains all cold retain variables/parameters of the application.
- The DiffCRValues file contains the cold retain variables/parameters where the initial value and the communicated value are different.
- The RValues file contains all retain variables/parameters of the application.
- The DiffRValues file contains the retain variables/parameters where the initial value and the communicated value are different.
- The CRCandidates file contains the retain variables/parameters where the initial value and the communicated value are different and variables/parameters are not written by 1131 code.

The content of a value file can be displayed in Microsoft Excel by opening the file with the Text Import Wizard. Select the InitalValue and CommValue columns and

set the data format to text as shown in [Figure 187](#).

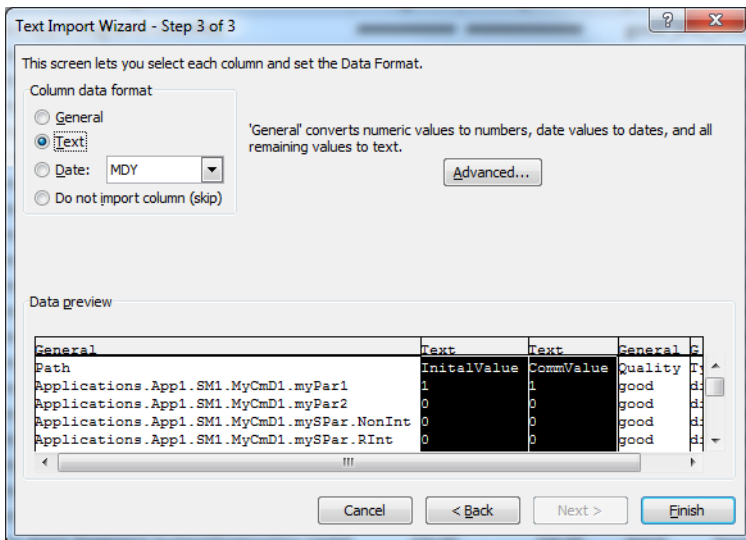


Figure 187. Text Import Wizard

Controller Configuration

Controller configuration includes the configuration of the controller Error Handler. The Error Handler is used to configure controller behavior on system errors. The following functions are provided by the Error Handler:

- General interface to report errors from all different parts of the system.
- Report mechanism to the operator for all types of errors and warnings.
- Handles system actions to, for example, stop the controller.

The majority of the reported errors can only be triggered internally by the system, and are not easily repeatable by the end-user. The Error Handler is not intended to be used for system diagnostics.

The Error Handler has no knowledge about redundancy, but redundancy may maintain the function of the system. If redundancy maintains the system function,

the reported error gets auto-acknowledged and for example, System Alarm Output gets in-activated.



The Severity in the Controller Settings editor is only applicable to the severity of the call to Error Handler, and should not be mixed up with the Severity of the System Alarm. See [General status bit ErrorsAndWarnings](#) on page 560.

Error Handler settings are made for each controller, in the Controller Settings dialog. There are certain settings that cannot be changed (they are dimmed in the dialog). You can add additional actions, but you cannot change the original settings.

The system engineer performing the configuration must be aware of the impact of the enabling Error Handler actions. For example:

- Controller Shutdown action decreases the controller availability. Controller Shutdown sets the PM in empty controller mode and erases the application programs.
- Log action increases the amount of print-outs in the controller log, which possibly makes it difficult to find important and relevant information.
- Event action increases the amount of events in the operators Event List, which possibly makes it difficult to sort out relevant information and events may even be overwritten.



AC 800M High Integrity (HI) controllers have a number of settings that are not present in a non-HI controller, see [Controller Settings in High Integrity Controllers](#) on page 427.

Error Handler settings are slightly different for High Integrity and non-High Integrity controllers:

- [Controller Settings in Non-High Integrity Controllers](#) on page 424 describes how to configure the Error Handler in a non-High Integrity controller.
- [Controller Settings in High Integrity Controllers](#) on page 427 describes Error Handler settings that are specific to a High Integrity controller.



Errors can be reported from the code using the ErrorHandler function block type or the ErrorHandlerM control module type. Using these types, errors identified by the code can be handled in the same way as other errors. For more information on how to configure the ErrorHandler(M) types, see corresponding online help.



The ErrorHandler(M) types should be used with care, since they can be used to reset the controller.

Controller Settings in Non-High Integrity Controllers

Figure 188 shows the Controller Settings dialog for a non-High Integrity AC 800M controller. It is displayed by right-clicking the controller in Project Explorer and selecting **Properties > Controller Settings**.

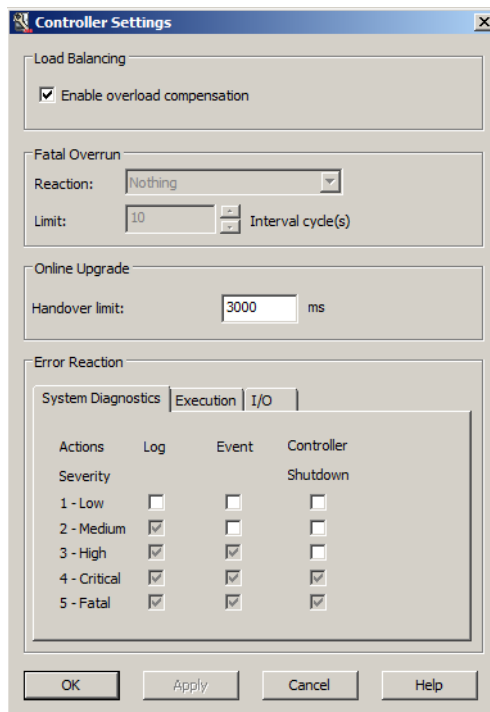


Figure 188. Controller Settings dialog for an AC 800M controller (non-HI).

If load balancing is enabled, overrun and latency supervision is automatically disabled, see [Overrun and Latency](#) on page 178.

The default setting for a non-High Integrity controller is that load balancing is enabled and overrun and latency supervision disabled. If you disable load balancing overrun and latency supervision is automatically enabled.

Fatal overrun settings are used only if overrun and latency supervision is enabled (this part will be dimmed if load balancing is enabled, see [Figure 188](#)).

The Fatal Overrun part of the dialog lets you set how many overruns (missed scans) that are allowed before a fatal error is considered to have occurred. The Reaction setting is used to select which action the controller should take when a fatal overrun error occurs. The options are Nothing, Stop Application, and Reset Controller (The default option is *Nothing*). The default setting for the Limit is 10 interval cycles.



It is important to avoid configuring the error handler in such a way that a fatal overrun error has two corresponding reactions, one that is set in the Fatal Overrun part of the dialog (for example, Stop Application) and one that is set in the Error Reaction dialog (for example, Controller Shutdown for the corresponding severity). Note that severity Fatal and Critical always lead to a controller shutdown.

If settings are inconsistent, you will receive a warning when trying to save the new settings.

For a non-High Integrity controller, the Error Reaction part lets the user set the following, see [Table 30](#).

Table 30. Error Reaction – non-High Integrity controller. This part of the dialog is used to set controller actions at system errors of different severity.

Severity	Log	Event	Controller Shutdown
1 Low	Configurable for all	Configurable for all	Configurable for all
2 Medium	Always for system diagnostics and execution Configurable for I/O	Configurable for all	Configurable for all
3 High	Always for system diagnostics and execution Configurable for I/O	Always for system diagnostics and execution Configurable for I/O	Configurable for all
4 Critical	Always	Always	Always
5 Fatal	Always	Always	Always

The above table shows controller reactions (fixed and configurable) when errors of different severities are received by the Error Handler in a non-High Integrity controller.

There are three different Error Types defined, and each Error Type may be configured with different actions for different Severities:

- **System Diagnostics:** General system errors for example, corrupt memory, full queues, lost communication reported by for example the logic solver or CEX module.
- **Execution:** Errors regarding IEC 61131-3 application execution, example latency, overrun, sequence verification, CRC (memory corruption), and so on. Execution Errors can also be activated from user defined diagnostics by using Function Block ErrorHandler or Control Module ErrorHandlerM.
- **I/O:** Errors from any Hardware Units or Modulebus scanner, for example channel error or faulty module.

Definition of Error Handler Severities:

- **1 Low**
Does not affect the system safety or the functionality of the reporting module.
- **2 Medium**
Does not affect the system safety but the functionality in the reporting module.
- **3 High**
May affect the system safety. The functionality in the reporting module is affected. Redundancy may maintain the function of the system.
- **4 Critical**
Affects the system safety; the whole reporting "subsystem" has failed. Redundancy may maintain the function of the system.
- **5 Fatal**
Unrecoverable software errors. The whole reporting subsystem has failed. Redundancy will not maintain the function of the system. This severity is only used when there is no possibility to safely continue using a back-up module.

Controller Settings in High Integrity Controllers

The Controller Settings dialog is different for an AC 800M High Integrity (HI) controller. It is displayed by right-clicking the (HI) controller in Project Explorer and selecting **Properties > Controller Settings**. There are also differences regarding what can be configured for the Error Handler, see [Figure 189](#).

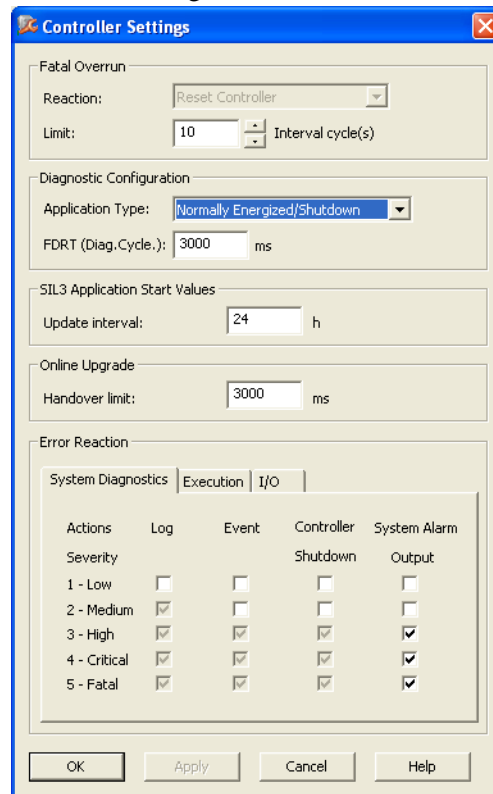


Figure 189. Controller Settings dialog for an AC 800M High Integrity controller.

Fatal Overrun settings are used to set how many overruns (missed scans) that are allowed before a fatal error is considered to have occurred. The Reaction setting is used to select which action the controller should take when a fatal overrun error occurs. The options are Nothing, Stop Application, and Reset Controller (The default option is *Nothing*). The default setting for the Limit is 10 interval cycles.



It is important to avoid configuring the error handler in such a way that a fatal overrun error has two corresponding reactions, one that is set in the Fatal Overrun part of the dialog (for example, Stop Application) and one that is set in the Error Reaction dialog (for example, Controller Shutdown for the corresponding severity). Note that severity Fatal, Critical and High always lead to a controller shutdown.

If settings are inconsistent, you will receive a warning.

The Diagnostic Configuration part of the dialog is only there if your controller is an AC 800M High Integrity controller. The Application Type setting affects the Error Handler Configuration options.

There are two possible values for Application Type:

- Normally Energized/Shutdown (default setting).
- Normally De-energized/Supervision which lets the user configure controller reset for system diagnostics and execution errors with severity High. Otherwise the settings are the same as for Low demand/Shutdown.

FDRT (Diagnostic Cycle Time) must contain a value that is 1000 or higher. The default value is 3000. Any value lower than 1000 is ignored. FDRT is the maximum elapsed time from the moment an error occurs, until action is taken. If FDRT is reached without any action being taken, an error with the severity Critical will be generated.

After a power fail the SIL3 applications are restarted using cold retain marked values which are periodically saved in the controller with a cycle time set by the user. The update interval can be set to a value between 1 hour and 24 hours to configure how often the values should be saved in the controller. Default value is 24 hours. See [Table 7](#).

For a High Integrity controller, the Error Reaction part lets the user set the following, see [Table 31](#).

Table 31. Error Reaction – High Integrity controller. This part of the dialog is used to set controller actions at system errors of different severity.

Severity	Log	Event	Controller Shutdown	System Alarm Output
1 Low	Configurable for all	Configurable for all	Configurable for all	Configurable for all
2 Medium	Always for system diagnostics and execution Configurable for I/O	Configurable for all	Configurable for all	Configurable for all
3 High	Always	Always	Always ⁽¹⁾	Configurable for all
4 Critical	Always	Always	Always	Configurable for all
5 Fatal	Always	Always	Always	Configurable for all

(1) If Application Type is set to Normally Energized/Shutdown, it is possible to configure controller shutdown for system diagnostics and execution errors with severity High.

The severities (left column in the table) and error types are the same as for non-High Integrity controllers, see page 426.

The System Alarm column in [Table 31](#), is only there for High Integrity controllers. If System Alarm is checked for a certain severity, a system alarm will be generated each time an error of the corresponding severity occurs.

Error Handler Log Entries

If an error of a certain severity is configured to be logged, it will generate a Controller System log (see [Controller System Log](#) on page 458) entry with the following general structure.

E yyyy-mm-dd hh:mm:ss:ms ErrorHandler PM: Error descr. (x,y,R)

- Such an entry should be read according to the [Table 32](#) table.

Table 32. How to read a log entry generated by the Error Handler.

Part	Description	Allowed Value(s)
E	Error	
yyyy-mm-dd	Date	
hh:mm:ss:ms	Time when error was time stamped	
ErrorHandler PM/SM:	Error detected by	ErrorHandler PM: =Processor Module ErrorHandler SM: Safety Module (HI controller only)
Error descr.	A text describing the error	
(x, y, ERS)	x=error type	1 (System Diagnostics) 2 (Execution), 3 (I/O)
	y=severity	1 (Low) 2 (Medium) 3 (High) 4 (Critical) 5 (Fatal)
	ERS=action type	E (Event) R (Reset) S (System Alarm)

Online Upgrade

Redundant AC 800M controllers can be upgraded with new firmware versions online. Online upgrade is initiated from Control Builder by a nine-step Wizard. Refer to *800xA Online upgrade and Co-existence, versions compatibility (3BSE080447)* in the ABB SolutionsBank for the supported upgrade paths.

Plan for Online Upgrade already at Project Design Phase

The key for success is to plan for online upgradeability already when making the initial installation. Certain design criterias exist and some consequences must be considered. It is recommended to document these for the future, so that the persons later on making an Online upgrade easily can analyze the process impact. One should, if possible, also perform an Online upgrade during Factory or Site acceptance tests, and document the result.

The following design criterias exists, and should be considered during design phase.

- The controller must be redundant, that is, two PM8xx forming a redundant pair.
- Define what Communication Interfaces that should be made redundant. Redundant CI units are not necessary; you can have a non-redundant CI unit in a redundant controller, but then the communication will be disturbed, and this will cause interrupted communication and may lead de-energized outputs. CI854 should, for example, be made redundant for this reason.



For more information about each CI unit's ability to support Online upgrade, refer to the *System 800xA System Guide Technical Data and Configuration (3BSE041434*)*.

- The process demands additionally 2 MB of free memory in the controller.
- Online upgrade cannot be performed for controllers with distributed applications.
- Use of the PPP protocol will prevent an Online upgrade.
- The Modulebus timeout on AI880, DI880 and DO880 must be set to at least four times the Modulebus Scan Cycle Time.
- SIL3 tasks may not use the option *Always Update Output Signal First in Next Execution*.

- They may be no unused tasks in the controller.
- The total controller load may not exceed 95 %.
- SM811 and SM812 for SIL3 application has a dedicated synchronization link to synchronize Active and redundant SM for hot-insert and Online upgrade. The synchronization link must be present during hot-insert and Online upgrade situations to copy data between two SM811s or two SM812s in a redundant setup.
- During the switchover of plant control, communication is interrupted for a short while. Communication blocks (example *MMSConnect* and *SBConnect*) may during this time indicate communication loss. Valid signals may drop for a few seconds. The application code must take this into account.



IAC (using communication variables) is not affected; its timeout during Online upgrade is automatically extended to 30000 ms or to the configured timeout value if that is longer, and this avoids any interruption in connection during the process.

- For Online upgrade when using Safe MMS communication, ensure that the *OLUTimeout* parameter is set to 30000 ms and that the *UseOLUTimeout* parameter is set to true in the MMSRead control modules. This avoids any interruption in connection during the whole Online upgrade process.
- The execution of tasks is halted for a couple of seconds during one of the upgrade steps. Define for how long the process can tolerate the controller to be frozen. Enter this value as the Handover Limit in the Controller Settings dialog. It is always advisable to choose the hand over time limit with a margin.



For High Integrity controllers, the Handover Limit time should not be set longer than the FDRT(Diag.Cycle.) setting.



The formula for calculating the actual time can be found in *System 800xA System Guide Technical Data and Configuration (3BSE041434*)*.

Why You Need to Read this First

Online upgrade is initiated from Control Builder by a 9-step Wizard that will guide you through the complete upgrading process. At a certain point during this process some actions will occur in the control system. The Wizard will for example temporarily stop the application, temporarily disconnect the redundancy and freeze the I/O update. Single CI units will be stopped during firmware download. All of these actions are harmless to your process – provided that you have prepared your Online upgrade in a correct manner!

In order to perform an Online upgrade successfully, you are strongly advised to start by acknowledging the preparations, requirements and prerequisites given in the sub-sections [Plan for Online Upgrade already at Project Design Phase](#) on page 431, [Restrictions for Online Upgrade](#) on page 434 and [Preliminary Actions for Online Upgrade](#) on page 435.

After that, proceed with the description in sub-section [Online Upgrade Process](#) on page 439, to fully comprehend the concept behind an Online upgrade process.

The last sub-section [Running Online Upgrade](#) on page 444, will help you to start the Wizard and begin an Online upgrade. You will be guided by a 9-step Wizard. And although these steps require a simple Next button click to proceed, some steps will contain additional buttons, which demands your special attention. For that reason, the Wizard has also been equipped with context sensitive Help buttons that will lead you directly to a descriptive topic page in Online help.

For example, the Wizard will sometimes during the process prompt you for different sub-actions inside a step which mean clicking buttons in a certain order of priority. When you enter a Wizard-step that contains multiple choices; click Help and follow the short suggestions for correct operations within that step.

Restrictions for Online Upgrade

Even though the Wizard will check the selected hardware configuration and analyze the applications status in the Project Explorer, you should confirm that your configuration is not affected by the following restrictions.



It is not allowed to make changes in the applications, or change any settings for controller or tasks before start of an Online upgrade session, except for changes caused by new library versions. No application is allowed to be added or deleted. If such changes have been made, a download is needed before start of the Online upgrade.



The Online upgrade is not allowed when the changes are downloaded from another engineering station.



Backup Media cards, that contain an application image or firmware files or have earlier contained firmware files and are not properly reformatted, will interfere with the Online upgrade process and must be removed from both the primary and backup PMs.

Technical Data and Performance

During the switchover of plant control, communication is interrupted for a while. Measurements have been done to exhibit the impact.

All the values listed below are typical values and they might vary from system to system:

- The trend values in Process Portal are interrupted for approximately 20 seconds.
- The alarms are delayed in the range of 20-30 seconds during switch of primary PM8xx.

Firmware Compatibility

The new versions of the firmware come from the new versions of the connected hardware libraries. The protocol handler in Control Builder analyzes the new firmware versions for compatibility.

Based on the compatibility check, the following settings are displayed during the upgrade process of the firmware in the hardware unit:

- *Mandatory*—This means that the unit must be upgraded with the new firmware due to compatibility reasons. It is not possible to deselect this option.
- *Recommended*—This means that it is possible to choose whether to upgrade or not by checking/unchecking the item. This item is checked, by default. If it is decided not to upgrade the item, the latest corrections made to the firmware will not be available, even though the firmware is compatible.
- *Not Available*—This appears when no valid upgrade is possible as no new firmware version is available.
- *Uncertain*—This appears when the old firmware versions in the primary unit and the backup unit are different. The protocol handler is unable to determine which firmware version is to be upgraded.



CI853 will always show up as changed/recommended, even though it is not changed. It is strongly recommended to de-select it, else will the serial communication be stopped unnecessarily.

Preliminary Actions for Online Upgrade

Put your process in a stable state, in which you can perform an Online upgrade with as little interference as possible. Basically, this means that you should try to identify a stage in the process where you will not receive alarm bursts, and where the process can handle a time interval of a few seconds without I/O communication and controller execution.

Use the 800xA System Status Viewer to make sure that all hardware units in the controller are fully operational and with no warning or errors, do not proceed otherwise.



Before performing an Online upgrade, ensure that all network cables are properly connected to the controller and the network is functional.

For controllers using a redundant network configuration, ensure that both primary and secondary network are operational before starting the Online upgrade sequence.

Both primary and backup PMs must be available on the network(s) during the Online upgrade process.

Online Upgrade of an AC 800M using CI857

Follow the guidelines below for a smooth Online upgrade of CI857 connected to the INSUM system:



Some of these guidelines are related to the nine steps in the Online upgrade Wizard in the Control Builder. See [Online Upgrade Process](#) on page 439.

- Set the parameter “FailSafe Heartbeat” on the INSUM TCP/IP Gateway to a value that is 1/4 of the shortest “Failsafe TimeOut” on the connected INSUM devices (MCUs and Circuit Breakers). Use the INSUM MMI or the INSUM OS to set this value to the parameter.
- During the Online upgrade, CI857 disconnects from the INSUM system for some time. Before CI857 disconnects from the INSUM system, it requests the INSUM TCP/IP Gateway to continue sending Failsafe Heartbeat to all INSUM subnets, until the CI857 reconnects.
- Since there is no redundancy for CI857, no commands can be sent to the INSUM system and the measurement values are not updated during this time.
- The duration of the broken connection between CI857 and the INSUM system depend on the upgradation of the firmware of CI857. If the firmware of CI857 is not upgraded, then it will be upgraded in the eighth step of the Online upgrade Wizard (together with the remaining units). See [Eighth Step – Upgrading Firmware in the Remaining Units](#) on page 444.
- After the Online upgrade, CI857 reconnects to the INSUM system and the communication is reestablished.
- If CI857 does not reconnect within the expected time, the INSUM TCP/IP Gateway stops sending Failsafe Heartbeat to the INSUM devices (MCUs and Circuit Breakers) and they go to Failsafe.
- During the switching in the seventh step, the status of the INSUMReceive and INSUMWrite blocks may be -5324 or -15 for up to 8 seconds. ProcessObjInsumLib takes care of this internally. See [Seventh Step – Switching the Process Control](#) on page 443.
- If an upgrade has been started and the communication between the CI857 and the INSUM TCP/IP Gateway is interrupted during the third step, where Redundancy is turned off, the upgrade will be terminated. See [Third Step –](#)

[Disabling the Redundancy](#) on page 441.

Expected Time for Online Upgrade of an AC 800M using CI857

The expected time for Online upgrade depends on the following factors:

- If the firmware is already up-to-date before the Online upgrade of the controller and an upgrade of CI857 is indicated as “Not Available” in the second step of the Online upgrade, the expected time is reduced. See [Second Step – Selecting Units to Upgrade](#) on page 441.
- If an upgrade of CI857 is indicated as “Recommended” in the second step of the Online upgrade, but the user decides not to upgrade, there will be a time delay of 15 seconds after the switching in step 7, after which the INSUM devices go to Failsafe.
- If an upgrade of CI857 is indicated as “Recommended” and the user decides to do the upgrade, the communication between CI857 and the INSUM system is broken two times. First, for some seconds during the seventh step, and later for a longer time during the eighth step while the firmware is downloaded.
- If the CI857 does not reconnect within 300 seconds after the start of the firmware download, the INSUM devices go to Failsafe.
- If an upgrade of CI857 is indicated as “Mandatory”, the communication between CI857 and the INSUM system is broken at the switching in the seventh step. This connection will not be reestablished until the upgrade of the firmware of CI857. If the CI857 does not reconnect within 900 seconds after the switching, the INSUM devices go to Failsafe. See [Seventh Step – Switching the Process Control](#) on page 443.

Online Upgrade of an AC 800M using CI858

Consider the following for the Online Upgrade of an AC 800M using CI858:



The version of CI868 Hardware Library connected in Control Builder project must always be greater than or equal to CI868 Hardware Library running in the CI868 Module.



It is mandatory to have only one CI868 Hardware Library under Control Builder Project Connected Libraries. That is, either 1.x or 2.x or 3.x CI868 hardware library version must be connected.



After CI868 Firmware upgrade, it is mandatory to re-import the SCD file to generate new CCF file compatible with new IEC 61850 stack.



During CI868 Firmware Upgrade, IEC 61850 Communication is interrupted until CI868 module is up and running.

Settings for High Integrity Controller

Access Enable

Enable the **Access Enable** switch in the controller to complete the Online upgrade.

Enable Extended Timeout for Safe Peer to Peer Clients

Enable the **Access Enable** switch in the controllers acting as Safe Peer to Peer Clients (Safe MMS) to the system, to extend the timeout handling in the communicating Control Modules.

Handover Limit Time

Before the Online upgrade, the Handover limit time in the High Integrity Controller must be specified in the Controller Settings dialog, and this setting must be downloaded to the controller. This is important because the time out cannot be adjusted without redoing the whole Online upgrade procedure.

Online Upgrade Process



All components of the controller must be fully operational before starting the Online Upgrade process and before performing the switch.

Even though this subsection is based on the Online upgrade Wizard, it merely describes the process in a conceptual manner. Do not try to run the Online upgrade Wizard solely on the basis of these steps.



It is strongly recommended to run the Wizard on a Control Builder node having direct connection to the Control Network. There should not be any routing between the Wizard and the controllers.

To get a complete and comprehensive guidance to a successful Online upgrade; study this conceptual explanation and then carefully follow the instructions given in the subsection [Running Online Upgrade](#) on page 444.

Primary, Backup and Trainee are three specific roles for redundant hardware units during the Online upgrade procedure. Consider the following to avoid any misunderstanding of the meaning for these roles:

- *Primary* is a process role responsible for executing the application(s).
- *Backup* is a process role responsible for maintaining redundancy. This means basically taking over the execution of a running application in case the Primary shuts-down.
- *Trainee* is a process role the backup unit enters after disabling redundancy in the Online upgrade.



If you decide to terminate an upgrade procedure by clicking **Cancel** in the Wizard, there are more or less serious consequences attached to that decision. Needless to say, performing an Online upgrade demands thorough planning before execution! There are some steps in the upgrade procedure that are more critical than others.

For example (step 4 and 7), interrupting the procedure after upgrading firmware in the fourth step will leave you with different firmware versions in the PMs, thus no possibility to easily regain redundancy. After such an interruption in the fourth step, you have to run Online upgrade again or do an Off line upgrade.



If the Online upgrade process should be interrupted or fail, always try to run the Wizard at least one more time.

If your upgrade process still fails, study the subsection [Solving an Interrupted Online Upgrade](#) on page 445.

The Online upgrade process can be performed using the nine steps in the Online Upgrade Wizard, as follows:

First Step – Analyzing the Project

The first step starts the Wizard and initiates the Online upgrade sequence. The Wizard begins by analyzing the project in the Control Builder and in the controller, and checks if an Online upgrade is possible. The analysis checks that:

- Hardware configuration is correct, a redundant controller with supported CI units only.
- All configured hardware units are available and functional.
- The applications in the redundant controller and in the Project Explorer are the same. Thus no additional changes in the Control Builder applications are permitted, besides connecting the new (hardware) libraries delivered with the 800xA for AC 800M system extension. See Restrictions for Online Upgrade on page 434.

Redundant controller:

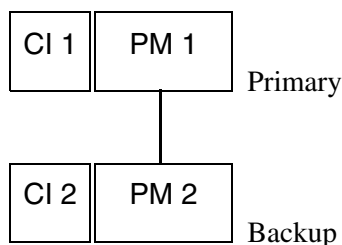


Figure 190. A schematic illustration of a redundant controller configuration before start of Online Upgrade.



The purpose of the figure above and figures in the following steps, is to show the components of a redundant controller and their different roles during the upgrade. Hence, the figure does not illustrate how to connect a redundant control system.

Second Step – Selecting Units to Upgrade

The Wizard performs a firmware version check on the redundant controller (PM and CI units) and another in the hardware libraries. An analysis compares the result of the read-outs and lists *mandatory firmware*, *recommended firmware* and *not available firmware* in a dialog box. The mandatory firmware must be upgraded, but firmware listed as recommended can be skipped in the upgrade, since the existing firmware is compatible with the new ones.

Third Step – Disabling the Redundancy

The Wizard lists which corresponding units that will be upgraded. The backup will be disconnected from the redundant controller and from now on it will be considered as a Trainee. After the backup PM is changed to a trainee its application will be removed and the PM 2 can no longer work as a backup. Thus the control system is no longer redundant, the system is singular.

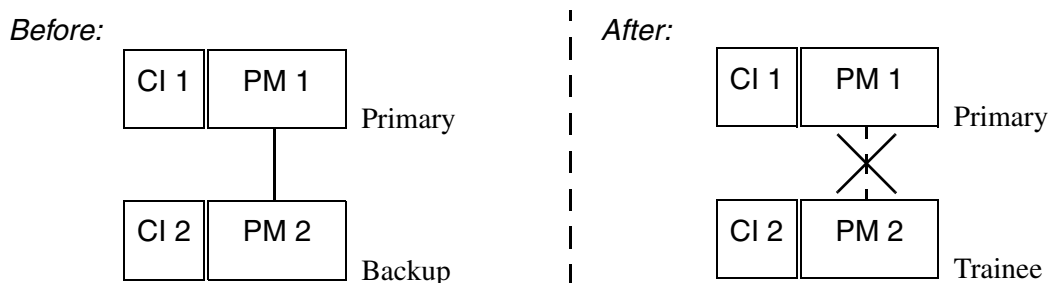


Figure 191. The redundancy is disabled and the backup is preparing for an upgrade. Thus the role switches from Backup to Trainee.



The redundancy line is marked with a cross to symbolize disabled redundancy. It does not imply disabled communication, as the RCU cable or BC810/BC820 is still physically attached. This means that PM 1 and PM 2 can still communicate).



During Online upgrade of an High Integrity Controller with SM811 or SM812 and SIL3 applications, a hot insert with activation is always needed either to regain redundancy after a successful Online upgrade, or if the redundancy has been broken during an unsuccessful Online upgrade attempt. Activate digital input 3 on the SM811 or SM812 after finishing the Online upgrade Wizard.

Fourth Step – Upgrading Firmware

The firmware is downloaded to the backup redundant CI units (if any) and to the Trainee.

This step provides three options:

1. Download firmware to the units and then wait until the upgrade is done (the Next button can be selected),
2. Remove the unit and download the firmware from another system and then insert the unit again.

A Refresh will verify that the upgraded units are physically present and have correct firmware versions. A typical user-case for this option is when a unit has been dismounted and upgraded elsewhere and then re-mounted afterwards again. Selecting refresh helps you proceed to the next upgrade step without downloading firmware a second time to the unit.

3. Proceed to the next step (only possible if all units are physically present and already have correct firmware versions).



Singular CI units will not be upgraded at this time. Instead they will be upgraded in step eight.

Fifth Step – Downloading Applications to the Trainee

Before the Control Builder downloads the application to the trainee the Difference Report is displayed. Study the report and decide whether to accept the difference report or not. It is completely normal that the report reflects different library versions. If you do not accept the difference report, you will return to step 5.

Sixth Step – Deciding the Online Upgrade Handover Limit

Although Online Upgrade Handover Limit is a quite simple timer function in Online upgrade, it still needs some considerations. At this stage the Primary is preparing to switch over control to the Trainee. It is a critical step for the controlled process, because during that time the task execution is halted. The Online Upgrade Handover

Time consists of time for stopping applications in the Primary, copying and transferring all application values, and handing over the control to the Trainee.



If the defined Online Upgrade Handover Limit time is exceeded, the Online upgrade procedure will be interrupted, and a roll-back of control to the Primary will take place. At the switch over control from Primary to the Trainee, a rollback will conclude within the specified handover limit time.

If the roll-back procedure fails, the I/O channels will immediately go to their Output Set as Predefined (OSP) values.

The suggested Online Upgrade Handover Limit in the Wizard is based on the value in the controller settings. The Online upgrade accepts an Online Upgrade Handover Limit of up to maximum 10 seconds (default is 3000 ms).



The Online Upgrade Handover Time is not equivalent to the total time that the I/O channel values are frozen. When the new Primary starts up, the Applications will start in the same way as at a “normal” warm restart after a download, for example. All tasks will be started according to their priority, interval, and Offset. This means that the Output freeze time is approximately equal to the Online Upgrade Handover Time, plus task offset and execution time for the first scan, plus delays incurred by higher-priority tasks.

Seventh Step – Switching the Process Control

Performing a switching means basically that the Online upgrade function sends a switch command to the Primary to switch over alarm states, variable values etc, to the Trainee and then resets itself. After the reset, the Trainee will exit its role and enter the Primary role and take over the process control.

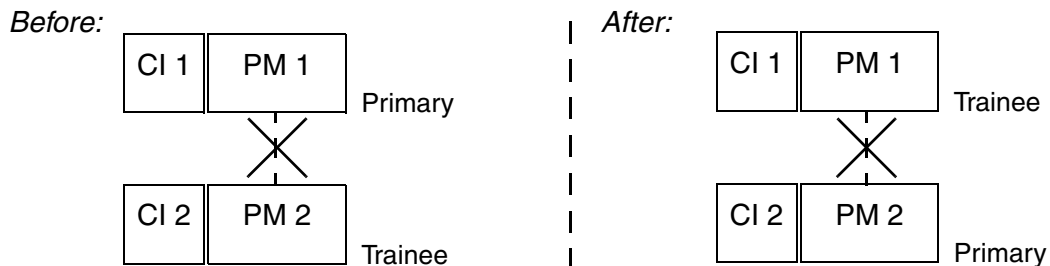


Figure 192. PM 1 and PM 2 are shifting roles. After this step the PM 2 will become the Primary and execute the running application.

Eighth Step – Upgrading Firmware in the Remaining Units

This step upgrades the remaining units in the redundant controller, which means any single CI units, the previous primary CI units and the old Primary PM. Similar to the previous forth step it provides three options, download firmware to the units and then wait until the upgrade is done (Next button can be selected). The Trainee resets and read the applications states from Primary. After that it becomes a backup and you should have full redundancy again.

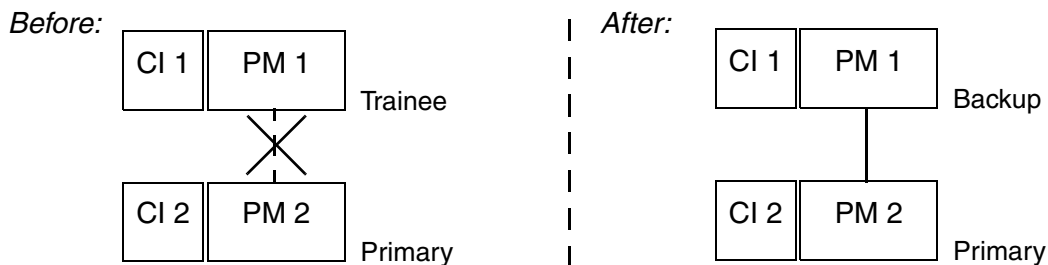


Figure 193. The redundancy has been re-established.

Ninth Step – Summarizing the Upgrade Process

The Wizard summarizes the results of the Online upgrade procedure. It is possible to save the summary by using the Save Summary button.

Running Online Upgrade

This subsection assumes that you have made all the necessary arrangements in your applications for an Online upgrade.

Make sure you have access to the Control Builder Online help. To make sure: On the **Help** menu, click **Help Topics**. Control Builder Online help Welcome page opens.



The Online upgrade function will perform all necessary controller resets automatically. Thus you should not try to perform any reset on the redundant controller while running an Online upgrade.

Starting the Online Upgrade Wizard

From the Project Explorer hardware tree:

1. Right-click the controller and select **Upgrade** from the context menu. The Online upgrade Wizard will open.
2. Click the **Help** button in the Wizard. An easy to follow 9-step instruction will open and guide you through the Online upgrade.

The Online upgrade instructions end here in the manual, and continue in the Online help!

Solving an Interrupted Online Upgrade

The solutions provided in this subsection assume that you have already tried to restart the Wizard but failed to continue the Online upgrade process.



A PM can be marked as *Uncertain*. This means that the Wizard cannot yet determine firmware version for that PM. If a PM is marked as Uncertain, it can either mean that the PM has been upgraded in another system previously or the Online upgrade was disrupted.

Error case handling different firmware versions in PM 1 and PM 2

The Online upgrade has been interrupted with failure, which implies that PM 1 and PM 2 have different firmware versions. The current situation is that one PM is running your process and the other one is not responding (dead).

Start by figuring out the cause for the interruption. A common cause is a too short Handover Limit time. Then restart the Wizard again and continue upgrading firmware in the remaining PM (the Wizard will automatically reset the dead PM).



Please note, this procedure has been changed since previous versions of this book.

Error case handling different firmware versions in CI 1 and CI 2

The Online upgrade has been interrupted with failure which implies that CI 1 and CI 2 have different firmware versions.

Start by correcting the PM firmware version according to the suggestions given above in (PM 1 and PM 2). Then restart the Wizard again and continue upgrading firmware in the remaining CI unit(s).

If the failure should still remain then you must down/upgrade the CI unit(s) in another control system (by Hot Insert).



Hot Insert or more precisely the insertion of a SM811 or SM812 into a running system affects SIL3 applications. The applications will be stopped while getting synchronized. When running SIL3 applications, the start of the synchronization must be accepted by the user, to configure the inserted module. This is performed by activating digital input 3 on the SM811 and SM812.

Error case handling when I/O Channels have returned to OSP

The Online upgrade has initially been interrupted due to time-out and then failed to rollback the application to Primary. The consequence is that the I/O channels have returned to OSP and neither one of the PMs are running the process. At this stage, there is no possibility for Online upgrade. You must start over by performing a complete reset on all units and then begin downloading firmware and applications again.

Trouble-Shooting

General

When a control system error occurs, it is important to investigate it as soon as possible. In doing this, the possibility of finding and eliminating the problem will be substantially increased. The reasons are:

- The personnel involved will not have forgotten what happened.
- The application software involved will not have been changed.
- The systems involved will not have been changed (location, setup etc.).
- You may need a work-around quickly, to be able to continue your work.

- Some errors only occur under very special circumstances and/or in special hardware/software configurations. The person who reports the problem may have the only installation/configuration where we know it could occur.



A well-described error, with all vital information included, will always increase the probability of correcting the error quickly and effectively. [Error Reports](#) on page 493 provides some hints when writing an error report.

The task of trouble-shooting is usually very difficult, and requires a great deal of intuition and ability to draw conclusions from known facts. This subsection aims to provide some guidelines on solving problems.

Here are some basic troubleshooting questions which should first be answered.

- What is the problem?
- Is it a known problem? Check the available information (for example, Release Notes and Product Bulletins) and discuss it with colleagues.
- Has the system worked previously (with the same hardware)? If so, the problem may have occurred due to poor installation or due to setup problems.
- Has anything been modified recently? The problem is often to be found in modifications. If possible, revert to the previous state, and test.
- Can the problem be linked to any special event?
- Is it possible to reproduce the problem?

Log Files

The Industrial IT products described in this subsection have built-in logging routines that continuously write to log files. Log files will contain important information whenever a failure occurs during a programming session, or when a controller is running. These files and the crash files (see section [Crash Dumps for Analysis and Fault-Localization](#) on page 467) are very useful for troubleshooting and contain crucial information for analyzing malfunctions.



If Control Builder is running on a terminal server the log files are saved in a particular folder for the used session. For further information, see [Running Control Builder on Terminal Server](#) on page 398.

System Log File

The *system log* is created the first time Control Builder is started (or if there is no log file), and is used to store general information concerning Control Builder. Examples of information logged are start/stop of Control Builder and changes in the setup of Control Builder via the **Tools** menu. The System log can be read via the menu entry **Tools > Maintenance > Analysis > System Log**. Figure 194 shows an example of the system log.

S	Date	Time	Category	User	EventDescription
I	2002-04-24	11:05:15	SYSTEMOP	Default	System log created at station 10.46.41.20
I	2002-04-24	11:05:15	SYSTEMOP	Default	Application is starting
I	2002-04-24	11:06:27	SYSTEMVAR	Manager	Value of system variable PromptCommentOn-Save manually changed to false

Figure 194. An example of the system log

The path and file name of the System log are given in Table 33.

Table 33. The System log file path.

Denomination	Path/Note
Control Builder M System Log	<p>Path / File name</p> <p>C:\ABB Industrial IT Data\Engineer IT Data⁽¹⁾\Control Builder M Professional \LogFiles\System.log</p> <p>Note</p> <p>Only one version of this file exists.</p>

(1) The default working directory is shown.

Session Log Files

At start-up, Control Builder, OPC Server for AC 800M, MMS Server for AC 800M, SoftController, and the Tool Routing Service for AC 800M, automatically creates a *session log* file on the hard disk. If the controller is a High Integrity controller, it also creates a Controller Configuration Integrity log. These files contain information generated during one session, that is, from the time the product is started, until it is stopped. New files will be created upon each new start-up.

At start-up, information about hardware and software versions, and later, information on system events, such as mode changes (Offline to Online, or vice versa) and error print-outs, will be logged in the session log. For High Integrity controllers, the Controller Configuration Integrity log will show the result of all test compilations that are made to make sure that the controller is not corrupted. Session logs are continuously updated in a running system, and whenever a problem occurs it is a good idea to look at the logs to see if there are any printouts. It is possible to read log files for the current session via the menus.



Session logs are saved from the previous nine sessions. It is important to save a file containing information about a problem, with a new name, before it is overwritten.

Ten successive start-ups will generate the following session log files; Session.log (from last start-up), Session.log _bak1 (next to last), Session.log _bak2, etc to Session.log _bak9 (the first start-up or oldest saved start-up). This means that when you start-up the system a eleventh time Session.log _bak9 will be overwritten and the previous Session.log will be renamed as Session.log _bak1 and a new Session.log will be created.



You will lose the oldest saved file because all the files are pushed one step after each start-up. This means that (_bak8) is pushed to (_bak9), (_bak7) to (_bak8) etc and Session.log to (_bak1).

- Session.LOG
- Session.LOG_bak1
- *Session.LOG_bakn.....*
- Session.LOG_bak9
- ~~Session.LOG_bak9~~

Below is an excerpt from Control Builder session log.

```
Starting MMS Server
MMS Server running
MMS Server connected

Product : Control Builder M Professional
Version : 3.1/0b3 (Build 0.44.2.2)
Created : 2002-04-16
ABB Automation Products AB

Working folder is: C:\ABB Industrial IT Data\Engineer IT Data\Control Builder M Professional 3.1
Microsoft Windows 2000 Professional version 5.0 Service Pack 2 (Build 2195)
Memory information
TotalPhys = 511 MB, AvailPhys = 347 MB
TotalPageFile = 1248 MB, AvailPageFile = 1081 MB
Heap size = 64 MB
Character set: West European/American
I 2002-04-19 19:33:22.570 MMS Server is running
I 2002-04-19 19:33:22.601 .
I 2002-04-19 19:33:22.633 Network address is 10.46.41.20, 172.16.84.163.
I 2002-04-19 19:33:23.914 Supported hardware definition syntax: 2.0
I 2002-04-19 19:33:23.914 Reading hardware definitions...
I 2002-04-19 19:33:26.023 342 hardware unit definitions created,
1371372 bytes heap used
I 2002-04-19 19:33:26.148 Mirror: No Aspect Server found. Mirroring not activated.
I 2002-04-19 19:33:26.914 License Manager: Hardware ID, 00-03-47-68-8E-A5 (Ethernet)
```

I: Information
W: Warning
E: Error

Figure 195. The first section of the Control Builder session log. Pay special attention to Warnings (W) and Errors (E)

The paths and file names of the session logs are given in [Table 34](#).

Table 34. Session log file paths.

Denomination	Path/Note
Control Builder M session log	<p>Path / File name C:\ABB Industrial IT Data\Engineer IT Data⁽¹⁾\Control Builder M Professional \LogFiles\Session.log</p> <p>Note Session log files stored from the last 9 sessions: Session.log Session.log_bak1, Session.log_bak2, <i>Session.log_bakn....</i> Session.log_bak9</p>
Controller Configuration Integrity Log	<p>(This log is only generated for High Integrity controllers.)</p> <p>Path / File name C:\ABB Industrial IT Data\Engineer IT Data⁽¹⁾\Control Builder M Professiona\LogFiles\CCI_Session.log</p> <p>Note Session log files stored from the last 9 sessions: CCI_Session.log CCI_Session.log_bak1, CCI_Session.log_bak2, <i>CCI_Session.log_bakn...</i> CCI_Session.log_bak9</p>
OPC Server session log	<p>Path / File name C:\ABB Industrial IT Data\Control IT Data⁽¹⁾\OPC Server for AC 800M\ LogFiles\Session.log</p> <p>Note Session log files are stored from the last 9 sessions: Session.log Session.log_bak1, Session.log_bak2, <i>Session.log_bakn....</i> Session.log_bak9</p>

Table 34. Session log file paths. (Continued)

Denomination	Path/Note
MMS Server session log	<p>Path / File name C:\ABB Industrial IT Data\Control IT Data⁽¹⁾\ MMS Server for AC 800M\ Session.log</p> <p>Note Session log files are stored from the last 9 sessions: Session.log Session.log_bak1, Session.log_bak2, <i>Session.log_bakn....</i> Session.log_bak9</p>
Tool Routing Service session log	<p>Path / File name C:\ABB Industrial IT Data\Control IT Data⁽¹⁾\ Tool Routing Service for AC 800M \ Session.log</p> <p>Note Session log files are stored from the last sessions: Session.log Session.log_Bak</p>
SoftController session log	<p>Path / File name C:\ABB Industrial IT Data\Control IT Data⁽¹⁾\ SoftController \ Session.log</p> <p>Note Session log files are stored from the last 9 sessions: Session.log Session.log_bak1, Session.log_bak2, <i>Session.log_bakn....</i> Session.log_bak9</p>

(1) The default working directory is shown.

OPC Server (Session.log) Example

The list example shows an extract from an OPC Server session log file and how to interpret the given data in four separate error occurrences. Important information has been highlighted with typeface bold.

E = error, **AE** = Alarm Event, **DA** = Data Access.

```
E 2003-11-07 11:11:54.867 On Unit= SubAlarmEvent ConnectionError-  
172.16.0.11 OPC Server (6500) Connection error to AE subscribed  
controller
```

```
E 2003-11-07 11:12:03.335 On Unit= SubDataAccess ConnectionError-  
172.16.0.11 OPC Server (5500) Connection error to DA subscribed  
controller
```

```
E 2003-11-07 11:12:04.913 Off Unit= SubAlarmEvent ConnectionError-  
172.16.0.11 OPC Server (6500) Connection error to AE subscribed  
controller
```

```
E 2003-11-07 11:12:27.398 Off Unit= SubDataAccess ConnectionError-  
172.16.0.11 OPC Server (5500) Connection error to DA subscribed  
controller
```

1. The first event description tells us that the OPC server lost connection (**On**) to controller for Alarm and Event subscription (and when this error occurred).
2. The second event description tells us that the OPC server also lost connection (**On**) to controller for Data and Access subscription.
3. The third event description tells us that the OPC server regained connection (**Off**) to controller for Alarm and Event subscription.
4. The fourth event description tells us that the OPC server regained connection (**Off**) to controller for Data and Access subscription.

As you can see, letter (E) stands for error and it occurs both when error activates (**On**) and when the same error is gone (**Off**).

Control Builder Start Log

Control Builder creates a *Start Log* file for logging the last Offline to Online transfer (in Test or Online mode). Information, such as warnings and error messages, will be logged. The Start log is very useful when investigating errors that might occur during or just after an Offline -> Online transition. Sometimes the Start log will give a natural explanation of what at first looks like an error (for example, lost Cold Retain values).

The nine latest Start logs are saved.



It is important to save a file containing information about a problem, with a new name before it is overwritten. Furthermore, check that the date and time in the Start log correspond with the time when the problem occurred.

The path and file name of the Control Builder start log, are given in [Table 35](#).

Table 35. The Control Builder start log file path.

Denomination	Path/Note
Control Builder M Start log	<p>Path / File name C:\ABB Industrial IT Data\Engineer IT Data⁽¹⁾\Control Builder M Professional\LogFiles\startlog.txt</p> <p>Note The nine latest Start log files are saved: startlog.txt startlog.txt_bak1, startlog.txt_bak2, <i>startlog.txt_bakn...</i> startlog.txt_bak9</p>

(1) The default working directory is shown.

Field Bus Parameter Log Files

During compilation and simulation, CI851, CI854 and CI860 master parameters will be automatically calculated.

The calculation is performed for all controllers in the project and for all masters connected to the controllers. The result is sent to text files, which is stored in the same place as the Control Builder log files. The text files have no backup, and are replaced at every compilation and simulation.

The path and file name of the Field bus parameter log files, are given in [Table 36](#).

Table 36. The Field bus parameter log files path.

Denomination	Path/Note
CI854 parameter log file	Path / File name C:\ABB Industrial IT Data\Engineer IT Data ⁽¹⁾ \ Control Builder M Professional\LogFiles\Profibus_DPV1_Calculation.txt
CI860 parameter log file	Path / File name C:\ABB Industrial IT Data\Engineer IT Data ⁽¹⁾ \ Control Builder M Professional\LogFiles\FF_HSE_Calculation.txt

Device Import Wizard Log File

When Device Import wizard is used a log file is created. If any failure during the import occurs, errors and/or warnings are written to the log file, together with a text describing the error/warning.

For a successful creation of a hardware definition file the log file contains some entries: date and time of use, version of wizard and parser component, contents of the device description file and contents of the generated hardware definition file.

When the file size of a log file reaches 10MB it will be renamed next time the Device Import Wizard is invoked and a new log file is created. If there are an backup file at that time, it will be deleted.

Table 37. The Device Import Wizard log file path

Denomination	Path/Note
Device Import Wizard log file	Path / File name C:\ABB Industrial IT Data\Engineer IT Data ⁽¹⁾ \ Control Builder M Professional\LogFiles\DIW.log

(1) The default working directory is shown

PROFINET configuration log file

The Control Builder creates a log file **PROFINET_Configuration.txt** during download. This log file will have the result of the download compilation for the current and previous configurations. The log file can store data upto 10 MB and is stored in the LogFiles directory in Control Builder. The current compilation result is stored at the end of the log file.

If the log file exceeds the maximum size of 10 MB, then the file is automatically saved as **PROFINET_Configuration1.txt** and a new **PROFINET_Configuration.txt** is created. A maximum of nine old log files will be saved before the oldest file gets overwritten. The log file also contains internally calculated data that are not available in the Control Builder.

Table 38. The Device Import Wizard log file path

Denomination	Path/Note
PROFINET Configuration log file	Path / File name C:\ABB Industrial IT Data\Engineer IT Data ⁽¹⁾ \Control Builder M Professional\LogFiles\PROFINET_Configuration.txt

(1) The default working directory is shown

Control Builder System Information Report

The *system information report* is a list of hardware, software and setup information for an engineering station. This information is generated by a menu command and presented in a text editor.

To generate a new report perform either of these two alternatives.

- Select menu **Help > About Control Builder M > List all Information**
- In the Control Builder Setup Wizard, click **Show Settings** button.
This alternative generates almost the same information as the alternative above, but fewer Environment variables are printed.



It is important to generate a new file containing information that was valid at the time the problem occurred.

The path and file name of the Control Builder M System information report file are shown in [Table 39](#).

Table 39. The Control Builder system information report file path.

Denomination	Path/Note
Control Builder M System information report	Path / File name C:\ABB Industrial IT Data\Engineer IT Data ⁽¹⁾ \Control Builder M Professional\LogFiles\SystemInformation.txt

(1) The default working directory is shown.

Heap Statistics Log

There is *heap statistics* log file for SoftController. Every time a message “memory full” occurs (see [Figure 196](#)) in these products, the system software will automatically generate a *heap statistics log* file containing information about the content of the **heap**¹.

If “memory full” occurs in a situation that cannot be explained as normal, then this file should be included in an error report to your supplier’s service department.

When a system is unable to store more information in the heap, an error message will be displayed. In most cases (more than 98%), this is due to an attempt to store too much information in too small a heap. If this occurs for a product running on an engineering station, increase the heap size for that product, using the Setup Wizard.

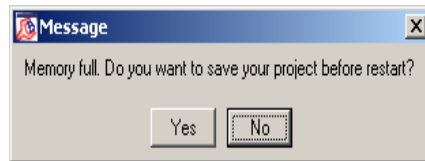


Figure 196. The “memory full” message.

The paths and file names of the *heap statistics* log files are given in [Table 40](#)

Table 40. The *heap statistics log file path*

Denomination	Path/Note
SoftController heap statistics log	<p>Path / File name C:\ABB Industrial IT Data\Control IT Data⁽¹⁾\SoftController \heapstat.dat</p> <p>Note The file is intended to be stored and included in an error report.</p>

Controller System Log

Controllers have a circular log buffer that can hold a certain amount of information, normally all information that has been generated during the last 5 to 8 start-ups.

1. A product, for example, a soft controller, uses a general memory area to store information. This area is called a **heap**. In the engineering station this area does not necessarily reside in the RAM memory.

A lot of the information gathered in a controller log file can be of great assistance, but a controller file is circular, which means that the last error often disguises more important previous errors. This means that the original error can be hard to discover. Therefore, you are advised to **first save the log file to a safe location** (no risk of deleting history) and then fault-find your way back. After renaming the first controller log file, it is safe to fetch as many controller log files as necessary.

The Controller System log is never deleted. Provided that the battery backup is working properly, the information can be retained during a power failure. This function makes it possible to restart a faulty system immediately to regain control of the process, without losing vital information about the error.



You must first save the Controller system log file on a safe location before fault-finding; it is much more difficult to identifying the original error after several startups.

The recommended way to access the Controller System log information is to fetch it via Control Builder. Selecting **Tools > Maintenance > Remote System...** will show a Remote System dialog, see [Figure 197](#).

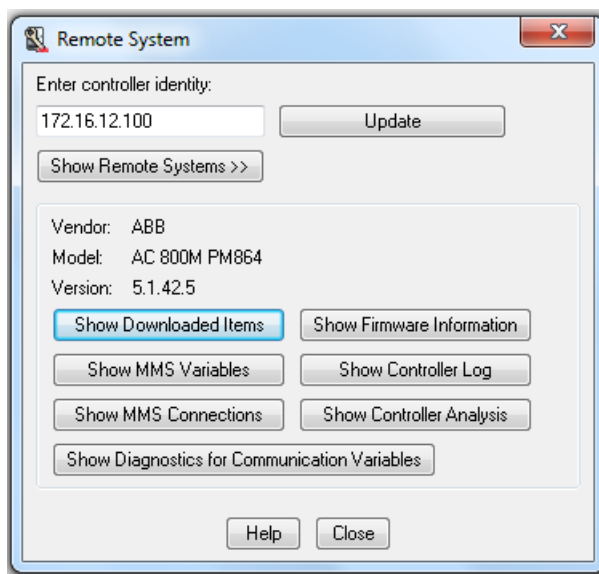


Figure 197. The Remote System dialog box.

Enter the controller identity (the IP address) and click on the **Show Controller Log** button to show the Controller System Log.



A redundant controller creates one log file for the primary unit and one for the backup unit, hence two different log files.

The information will be shown in a text editor and also be stored in a file.

However, the first controller log can still be overwritten. The 'First-in-First-out' principle is still valid for controller logs if you activate the 'Show Controller Log' function from the Project Explorer.

Figure 198 below, is an excerpt of the controller system log.

```
Product : AC 800M PM864
Version : 5.1.0/0 (Build 5.1.42.5)
Created : 2010-01-20
ABB AB

Controller Reset
Position Module      Firmware Name      Date      Version
0          PM864      Fw864             2010-01-20 5.1.42.5
              OMEGA             2010-01-20 5.1.42.5
              CPU Unit          N/A          PM864A D
              Backplane        N/A          TP830 B
              FPGA           N/A          1.3/0
              CEX Master       N/A          2.3
              CEX Slave        N/A          1.1
              CPU chip      N/A          0700
              RCU          N/A          2.1

Actual heapsize: 25200 kbytes
```

Figure 198. One section of the controller system log showing the actual firmware in the controller.

The path and file name of the Controller System log file are given in Table 41.

Analyzing Controller Logs

During compile and download the logs are fetched, analyzed and saved if the controllers configuration is empty and the logs contains crashes.

If the logs contains crashes then they are saved with following names:

- Controller aa.bb.ccc.ddd yyyy-mm-dd-hh.mm.ss.ttt CPU.log
- Controller aa.bb.ccc.ddd yyyy-mm-dd-hh.mm.ss.ttt BackupCPUlog
- Controller aa.bb.ccc.ddd yyyy-mm-dd-hh.mm.ss.ttt CI.log

If controllers configuration is empty but none of the logs contain crash information, then the files are stored with following names indicating that this was a download to an empty controller:

Controller aa.bb.ccc.ddd (empty) CPU.log
Controller aa.bb.ccc.ddd (empty) BackupCPUlog
Controller aa.bb.ccc.ddd (empty) CI.log

aa.bb.ccc.ddd represents the IP-address.
yyyy-mm-dd-hh.mm.ss.ttt represents the current date and time.

The analysis is also done when logs are fetched through Remote System dialog. If the logs contains crashes the file name includes date and time. If crashes are not found in the logs, then they are saved with following default names:

Controller aa.bb.ccc.ddd CPU.log
Controller aa.bb.ccc.ddd BackupCPU.log
Controller aa.bb.ccc.ddd CI.log

aa.bb.ccc.ddd represents the IP-address.

Controller Logs Sent to Computers at Shutdown of Controller

At a controller shutdown, the Controller System log is automatically sent out on the Control Network as a broadcast message. It is fetched and stored in the working folder for the MMS Server on all computers running an MMS Server.



If the Controller System log, fetched via the Remote System dialog, after a shutdown is empty due to a battery failure in the controller, the log will still be present at all computers running an MMS Server. It is then possible to find it in the following path:

C:\ABB Industrial IT Data\Control IT Data\MMS Server for AC 800M\
Controller aa.bb.ccc.ddd CPU.log

In this path you will also find the communication interface log file (Controller aa.bb.ccc.ddd CI.log).

See also [Figure 95](#) for dumps analysis due to controller/crash.

Table 41. The controller system log and communication interface log file paths .

Denomination	Path/Note
Controller System log Primary CPU	<p>Path / File name</p> <p>All controllers: C:\ABB Industrial IT Data\Engineer IT Data⁽¹⁾\Control Builder M Professional\LogFiles\Controller aa.bb.ccc.ddd CPU.log</p> <p>Note aa.bb.ccc.ddd is the IP address of the controller. See Controller System Log on page 458. The nine latest Controller System logs are saved: Controller aa.bb.ccc.ddd CPU.log Controller aa.bb.ccc.ddd CPU.log_bak1, Controller aa.bb.ccc.ddd CPU.log_bak2, etc Controller aa.bb.ccc.ddd CPU.log_bak9</p>
Controller System log Backup CPU	<p>Path / File name</p> <p>All controllers: C:\ABB Industrial IT Data\Engineer IT Data⁽²⁾\Control Builder M Professional\LogFiles\Controller aa.bb.ccc.ddd BackupCPU.log</p> <p>Note aa.bb.ccc.ddd is the IP address of the controller. See Controller System Log on page 458. The nine latest Controller System logs are saved: Controller aa.bb.ccc.ddd BackupCPU.log Controller aa.bb.ccc.ddd BackupCPU.log_bak1, Controller aa.bb.ccc.ddd BackupCPU.log_bak2, etc Controller aa.bb.ccc.ddd BackupCPU.log_bak9</p>

Table 41. The controller system log and communication interface log file paths (Continued).

Denomination	Path/Note
Communication Interface log	<p>Path / File name</p> <p>All controllers: C:\ABB Industrial IT Data\Engineer IT Data⁽¹⁾\Control Builder M Professional\LogFiles\Controller aa.bb.ccc.ddd CI.log</p> <p>Note aa.bb.ccc.ddd is the IP address of the controller. See Controller System Log on page 458. The nine latest Communication Interface logs are saved: Controller aa.bb.ccc.ddd CI.log Controller aa.bb.ccc.ddd CI.log_bak1, Controller aa.bb.ccc.ddd CI.log_bak2, etc Controller aa.bb.ccc.ddd CI.log_bak9</p>

(1) The default working directory is shown.

(2) The default working directory is shown.



The Communication Interface log (Example, the log in SM810, CI867 and CI868) is not battery protected. Hence, the log will be erased when the power to the controller is cut. The log includes vital information after a controller shutdown due to safety measures e.g. task latency etc. It is important to restart the AC 800M HI controller by pressing the INIT button, because this will preserve the log. Note that any attempt of restarting the AC 800M HI controller by toggling power will erase the log.

Fingerprint tool

The Fingerprint tool can be used online for collecting data, diagnostics, information about settings, versions, log files and so on, from all AC 800M controllers within a running system. The result can be used both prior to an upgrade to compare used product versions and firmware with Release Notes and Field Alerts and for troubleshooting. It will only take some minutes to run the tool also in large systems and the result will be stored in a directory created by the tool itself.

The tool creates a new directory each time the tool is executed as in [Figure 199](#). The directory name is unique because it contains the time when the tool is started. The

directory contains up to ten files per controller. These files begin with the IP address for each controller and do also contains its controller name. In addition the directory contains two files named Fingerprint.txt and SessionLog.txt. as in [Figure 200](#).

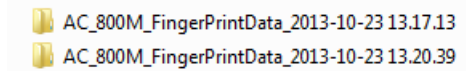


Figure 199. Directories created by the Fingerprint tool

There is one FingerPrint.txt file for each controller and one sum file containing the information for all controllers according to [Figure 200](#).

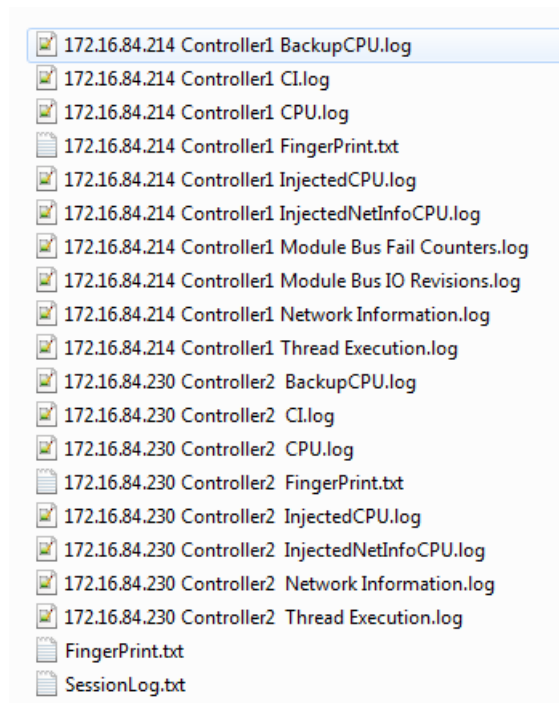


Figure 200. Files in the Fingerprint Directory

If the controller is empty then the name EmptyController plus IP address will be created.

The entire content of FingerPrint.txt files could be imported into Excel to make it readable and used by other applications. It can also be imported to other text handling tools for further filtration and analysis.

The Fingerprint tool can be opened from Engineering & Development/Control Builder MTools\Fingerprint in installation media. The tool is also included in the MMS Server for AC 800M installation.

An alternative option is to open the tool from a DOS prompt as in [Table 42](#) and specify which AC 800M controllers to collect data from by typing their IP addresses after the tool name, for example, `AC800Mfingerprint.exe 172.16.80.150`, or to create a list of IP-addresses as arguments to the console application. If no argument is specified, RNRP is used, else the provided list is used. The tool communicates with the controllers in parallel, in order to gain performance.



It is not allowed to simultaneously run the tool and perform downloads to the AC 800M controller(s). Therefore, it is not recommended to run the tool cyclically either.

Table 42. Fingerprint Tool from DOS Prompt

Nr	Command line	Description
1	AC800MFingerprint	The tool will use the RNRP in order to get the IP-addresses of all AC 800M controllers in the network. The tool will operate in full fingerprint.
2	AC800MFingerprint 172.16.85.128 172.16.85.3 172.16.85.7	The tool will only communicate with the controllers with these addresses. The tool will operate in full fingerprint.

Table 42. Fingerprint Tool from DOS Prompt

Nr	Command line	Description
3	AC800MFingerprint wd="C:\Temp	As number 1 but the working directory is set to C:\Temp.
4	AC800MFingerprint wd="C:\Temp" 172.16.85.128 172.16.85.3 172.16.85.7	As number 2 but the working directory is set to C:\Temp.

System versions from SV5.0 SP2 and forward are supported. RNRP has to be installed on the PC. Supported operating systems are: Windows 8, Windows 7, Windows XP, Windows Server 2012, Windows Server 2008 and Windows Server 2003. Controller versions from SV5.0 SP2 and forward are supported. The tool uses MMS and it uses the same communication services as the Control Builder.

The following data is collected from each controller:

- Version analysis data of the controller and the applications
- Firmware version information of the CPU and CI modules (For use prior to upgrade or analysis.)
- Modulebus IO Revisions (For use prior to upgrade or analysis)
- Modulebus Fail counters (Information for IO problem analysis and so on)
- Clock synch status (SynchProtocol, ClockMasterStatus, TimeQuality and so on)
- CPU load (To see if the controller is running with high CPU load.)
- CPU runtime (As reference to counter values etc during analysis.)
- Heap utilization (Used CPU memory, heap leakage analysis and so on)
- MMS Load
- MMS diagnostics (NrOfMMSConnections, NrTransactionsPerSec and so on)
- IAC diagnostics (Only available for SV 5.1 and later controllers)
- Thread execution information (Information for controller problem analysis and so on.)

- Tasks information (Only available for SV 6.0 and later controllers)
- CPU log, CI log and Backup CPU log (Also stored before any counters overwrites the log.)
- Network information (Counters showing network information for analysis.)
- Dynamic protocol handler information

Crash Dumps for Analysis and Fault-Localization

If a crash occurs (in Control Builder, OPC Server, SoftController, MMS Server for AC 800M, or the Tool Routing Service for AC 800M), two new files are generated at the same location as the session log files. The first one is a dump file and the second is a rewritten session log file. These two files contain crucial information that should be delivered to the support personnel.

If a shutdown/crash occurs in an AC 800M Controller, a Post Mortem Memory Image may be saved on a Backup Media card inserted in the controller. See also [Dump of Post Mortem Memory Image](#) on page 220.

If a Control Builder crash occurs at 16:20 on the 19:th of May, then a dump file and a rewritten session log file will look like:

ControlBuilderPro 2006-05-19 16.20.29.184.dmp

ControlBuilderPro 2006-05-19 16.20.29.184 Session.LOG

Remote Systems Information

A connected remote control system¹ can be inspected and maintained from Control Builder. This can be an important tool when troubleshooting the system.

Select **Tools > Maintenance > Remote System** to open the Remote System dialog, see [Figure 201](#).

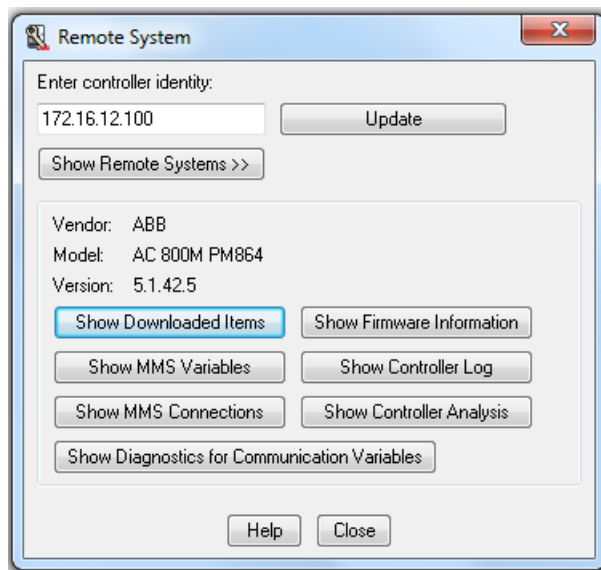


Figure 201. Remote System dialog.



The “Show Remote System” function can only list nodes on the same physical network! Thus, you must connect a Control Builder PC on the same Ethernet network; you cannot Show Remote System on nodes beyond routers, sub-networks etc.

1. Remote systems are controllers, OPC servers, and engineering stations connected to the same Control network as your own local system.

The following remote system functions are available, see the [Table 43](#) below. Click on a button in the dialog to retrieve information.

Table 43. The available remote system dialog functions.

Menu Item	Function
Show Remote Systems	Shows a list of all addresses to the control systems (including MMS process numbers) connected to the same network as the requesting system.
Show Downloaded Items	Shows information about controller configuration and about the application(s) running in the selected remote controller system, such as application name, application status, compilation date and time, compiling engineering station identity, and the checksum of the application. You can also remove a running application here. You can also access the source code report from the Show Downloaded Items dialog, see Source Code Report Generated for Project in the <i>System 800xA Control AC 800M Getting Started (3BSE041880*)</i> .
Show Firmware Information	Shows information from a controller, such as unit position, type of hardware unit, name and version of the current firmware and firmware creation date. Firmware can also be loaded to selected controllers here.
Show MMS Variables	Shows all the MMS variables in the system.
Show Controller Log	Shows the Controller System log, described in the section Controller System Log on page 458.
Show MMS Connections	Shows connection information about the remote systems, such as IP address, server/client function, identity of the connected system (destination system), usage, and number and maximum of transactions sent since connection was established.

Table 43. The available remote system dialog functions. (Continued)

Menu Item	Function
Show Controller Analysis	<p>Shows the Controller Analysis dialog that is used to:</p> <ul style="list-style-type: none"> • Reset the Module Bus Fail Counters in the selected controller. • Get the selected result/data from the controller. <p>The user can obtain results for "Heap Statistics", "Module Bus Fail Counters", "Module Bus I/O Revisions", "Network Information" and "Thread Execution".</p> <p>The respective result, obtained from the controller log, is saved to a new log file. The file name of the new log contains the "Controller ID" and the selected result. For example, 172.16.85.187_Heap Statistics.log.</p>
Show Diagnostics for Communication Variables	<p>Shows a diagnostic overview of the internal and external communication using the communication variables in the controller. For details, see Diagnostics for Communication Variables on page 471.</p> <p>The dialog displays:</p> <ul style="list-style-type: none"> • Unresolved communication variables • Different counters for errors/warnings, cycle times, and timeout. • Details about variable transaction in each server connection and client connection. • Out variables of the selected node.



For further information, refer to Control Builder online help. Use the Help button in the Remote System dialog, see [Figure 201](#).

Diagnostics for Communication Variables

The diagnostics tool for communication variables can be launched from the Remote System dialog of the selected controller. Click **Show Diagnostics for Communication Variables** in the Remote System dialog.

The first window that appears is the overview window. This window is a modeless window, that is, it is possible to bring up and work in other windows in parallel.

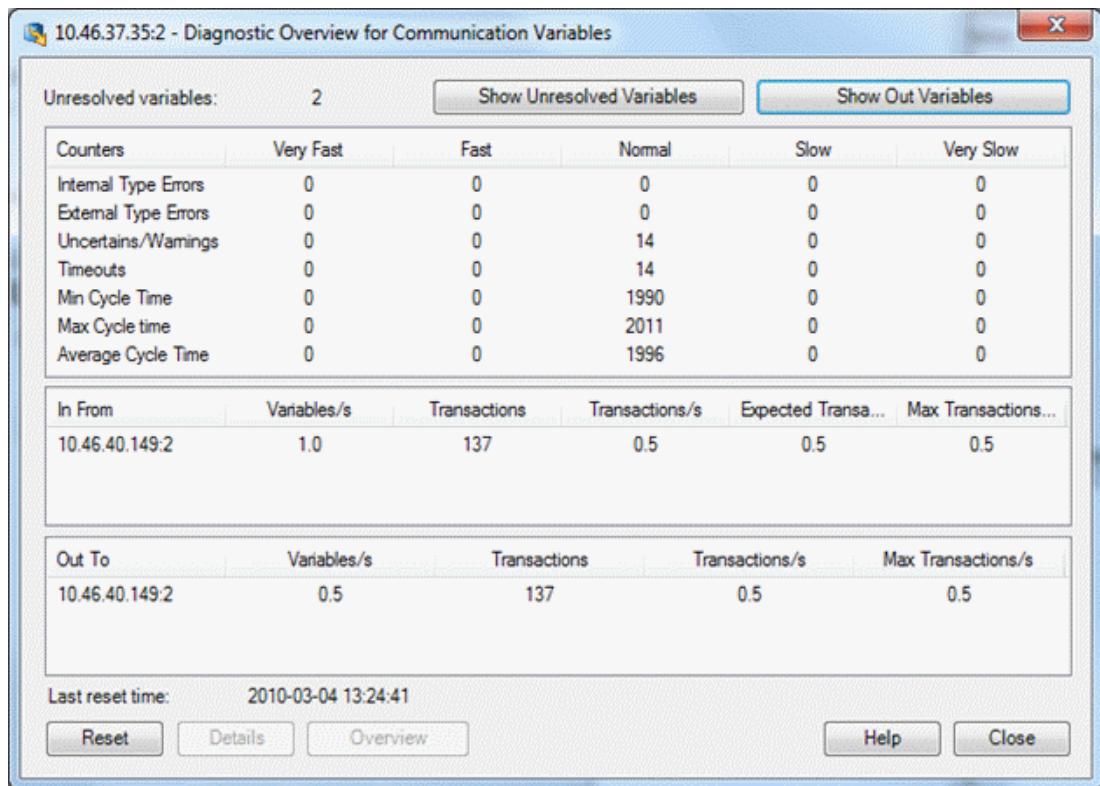


Figure 202. Diagnostic Overview for Communication Variables dialog

The Diagnostic Overview for Communication Variables dialog contains three panes that display information about the communication variables that are communicating through the applications in the selected controller.

The information is cyclically updated. The update interval is set to 5 seconds.

Counters Pane

The first pane lists different counters for the communication variables in the controller. Each column in the pane corresponds to a cycle time category of the communication variables.

The counters display the following values corresponding to the cycle time category in different columns:

- Internal type errors–Type mismatch during communication between applications within this controller.
- External type errors–Type mismatch during communication with an application in another controller.
- Uncertains/Warnings–Variables that are not updated within the requested time interval.
- Timeouts–Variables that are not updated within the requested timeout interval.
- Min Cycle Time–The lowest detected cycle time.
- Max Cycle Time–The highest detected cycle time.
- Average Cycle Time–The average cycle time.

In From and Out To Panes

The second pane "In From" contains information about the external client connections with respect to communication variables in the selected controller. The third pane "Out To" contains information about the external server connections with respect to communication variables in the selected controller.

Each pane contains columns for:

- Variables/s–An average value, calculated since last reset time.
- Transactions–Number of transactions since last reset time.
- Transactions/s–An average value, calculated since last reset time.
- Max. Transactions/s–Maximum number of transactions per second since last reset time.

- **Expected Transactions/s**—Number of expected transactions per second from the client. This column appears in the In From pane only.

The *Last reset time* shows the time when the Reset button was pressed.

Buttons

There are seven buttons in the overview:

- **Show Unresolved Variables**—Click to open the Unresolved Variables dialog.
- **Show Out Variables**—Click to open the Out Variables dialog.
- **Reset**—Click to reset the information in the controller. New values will be fetched.
- **Details**—Click to open the Detailed Diagnostics dialog for the selected client connection.
- **Overview**—Click to open the Diagnostic Overview for the selected server or client connection.
- **Help**—Click to open the online help topic for the diagnostic tool.
- **Close**—Click to close the window.

Show Unresolved Variables

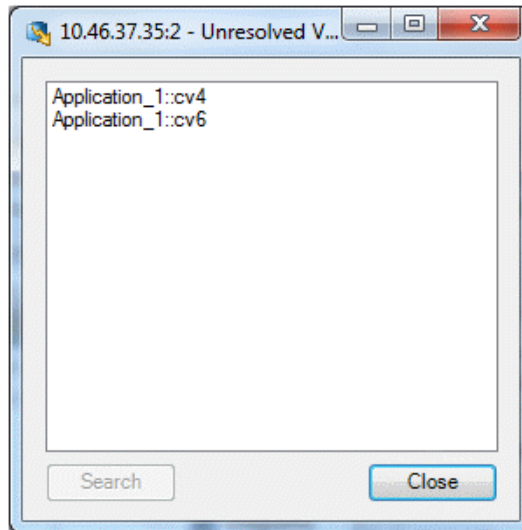


Figure 203. Unresolved Variables dialog

The Unresolved Variables dialog displays the list of unresolved variables. Select the variable and click **Search** to open the Search & Navigation tool for the selected variable.



The Search works in Offline mode and when the setting "Iterative Search in Online Mode" is set to false.

Show Out Variables

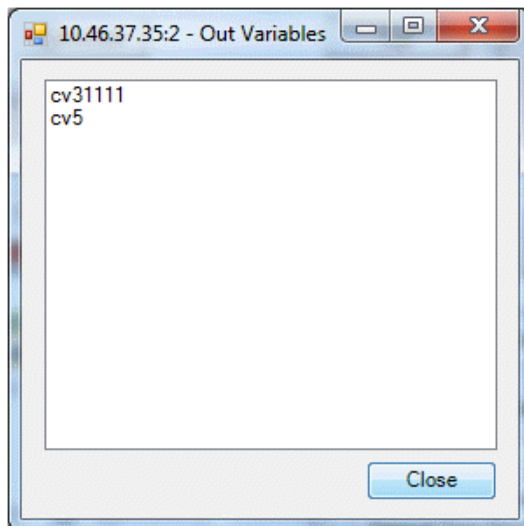
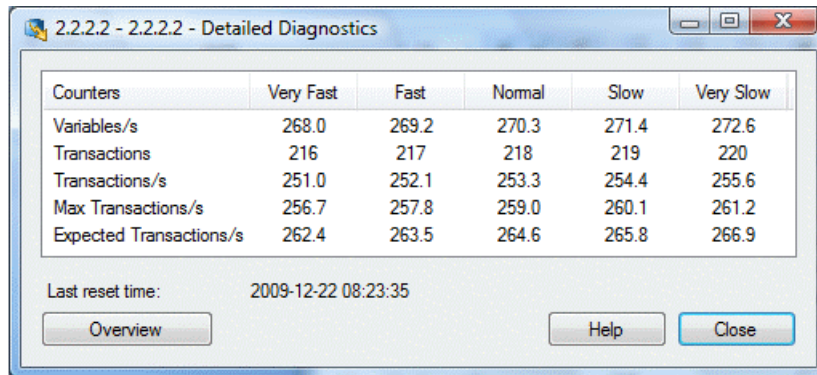


Figure 204. The Out Variables dialog

The Out Variables dialog displays the list of out variables in the controller.

Details



Counters	Very Fast	Fast	Normal	Slow	Very Slow
Variables/s	268.0	269.2	270.3	271.4	272.6
Transactions	216	217	218	219	220
Transactions/s	251.0	252.1	253.3	254.4	255.6
Max Transactions/s	256.7	257.8	259.0	260.1	261.2
Expected Transactions/s	262.4	263.5	264.6	265.8	266.9

Last reset time: 2009-12-22 08:23:35

Overview Help Close

Figure 205. Detailed Diagnostics dialog

The Detailed Diagnostics dialog displays the information for the different cycle time groups in a selected client connection. The following values are shown:

- Variables per second. An average value, calculated since last reset time.
- Number of transactions since last reset time.
- Transaction per second. An average value, calculated since last reset time.
- Maximum number of transactions per second since last reset time.
- Expected number of transaction per second. A value that is calculated at compile time.

Click **Overview** to open the Diagnostic Overview (Figure 202) for the controller that owns this client connection.

The information is cyclically updated. The interval is set to 5 seconds and it cannot be changed.

Analysis Tools

Control Builder Tools

The Control Builder **Tools** menu contains more useful tools for troubleshooting. Note that a great deal of the information is only valuable for your supplier's service department.

Select **Tools > Maintenance > Analysis** to open the following menu items, see [Table 44](#).



For further information, refer to Control Builder online help.

Table 44. The menu items of the Analysis tool.

Menu Item	Function
Disable Double-buffering	<i>Not useful for troubleshooting</i>
Disable Information Zoom	<i>Not useful for troubleshooting</i>
Disable Clipping	<i>Not useful for troubleshooting</i>
Image Selector Info in Online Mode	<i>Not useful for troubleshooting</i>
Image Selector Information	<i>Not useful for troubleshooting</i>
Show control modules in Online Mode	<i>Not useful for troubleshooting</i>
Write Variable Memory	<i>Used for counting modules and instances.</i>
Write Exported Variables	<i>Not useful for troubleshooting</i>
Write Variables in View	<i>Not useful for troubleshooting</i>
Start log	Shows the Control Builder Start log, described in Control Builder Start Log on page 454.
System log	Shows the Control Builder System log, described in System Log File on page 448.

Statistics for Application

The user can get the statistics about the application, for example, the number of instances that exists in the application. This is useful when the maximum number of instances has been exceeded. From the context menu of the selected application, select **Statistics** as shown in [Figure 206](#).

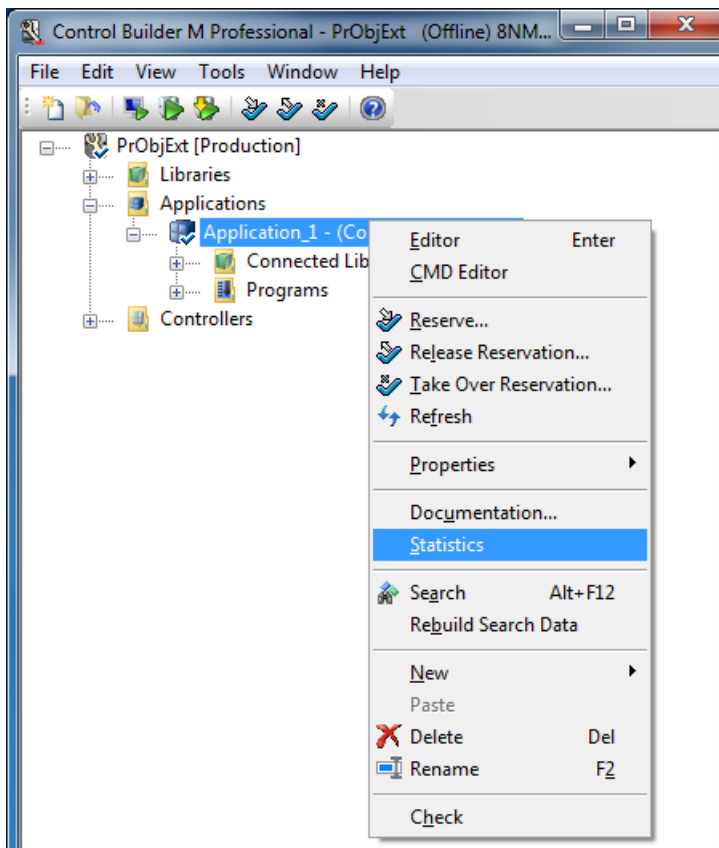


Figure 206. Obtaining statistics for the application

The maximum number of instances in an application is 65536. When this number is exceeded, the following dialog is shown.



Figure 207. Error message shown at download, when an application has too many instances.

System Diagnostics

System Diagnostics Function Block

The Basic library contains a function block type called *System Diagnostics*. You can use this function block type to measure and display the following functions.

- Cyclic load resulting from task execution,
- Stop time and memory usage during a controller download,

A stop at each download occurs when a new application version is downloaded. The length of this stop is mainly affected by the number of variables marked as retain or cold retain. Each variable marked retain or cold retain must keep the value at a download with warm restart. The copying takes time and copying more values requires more time. The attribute retain should only be set for variables if they really require such an attribute, as this attribute increases the stop time during download of program changes to the controller. When the new version of the application has been created in the controller, the old application is stopped (i.e. start of stop time) then all values are copied from the old application to the new. The new application version is then started (i.e. end of stop time). During the copying instances, the application exists in two versions, one old and one new. This is the reason for the extra memory needed.

- Current memory in use,
- Maximum memory used since the last cold start,

- Alarm and event information,
- Total CPU Load,
- Ethernet statistics:
 - number of data packages sent,
 - number of sent data packages that were lost,
 - number of data packages received,
 - number of received data packages that were lost.

The System Diagnostics function block is, as default, located in one of the *Diagram* folders of the Project Explorer tree, see [Figure 208](#).

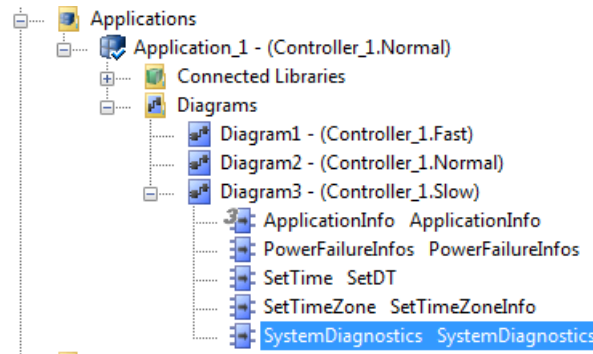


Figure 208. The System Diagnostics function block

Values can be updated either on command or cyclically using the Interaction Window, which is opened by selecting the *System Diagnostics* function block, right-clicking, and then selecting **Interaction Window**.



The System Diagnostics Interaction window is only available in Test/Online mode.

System Diagnostics Interaction Window.

The System Diagnostics Interaction window contains system memory and program download information. The interaction windows can be displayed in two versions, *Simple* and *Advanced*.



The values shown in Test mode are not those valid in Online mode. You cannot use this information to check in advance which controller size you have to purchase.

The *Simple* Interaction window contains the following information:

Table 45. The Simple Interaction window

Function	Description
System	Displays the TCP/IP address of the supervised system.
Cyclic load	Displays cyclic load due to task execution in percent.
Latest update	Displays the time of the last update.
Cyclic update	Cyclic update is activated by checking the check box. Cyclic update interval is set in time format, for example 5 m (5 minutes).
Total Load CPU	Shows the total CPU load for the controller. The total load is available as a parameter of type dint, called <i>TotalSystemLoadPerCent</i> .
Ethernet Statistics	By clicking the Ethernet button, you display Ethernet statistics in a separate window. This window shows the number of sent/received packages, and how many of those that were lost. These statistics are available as parameters. There are also parameters for resetting the counters. See online help for the SystemDiagnostics function block.

Click on the **Advanced** button, and the *Advanced* Interaction window will appear. It contains the following additional information.

Table 46. The Advanced interaction window.

Function	Description
Memory size	The allocated heap size, see Figure 209 .
Used memory	The part of the heap used in bytes and percent of the total heap size.
Max used memory	The maximum part of the heap used in bytes and percent of the total heap size.
Memory quota	The part of the total heap size available when program changes are sent to the controller. If the memory quota is exceeded an error icon is displayed. Note. This setting is only used for a warning indication.
Stop time	Stop time during the last download.
Init peak memory	Memory used during initiation phase.
Used memory at stop	The part of the heap used during the stop phase in bytes and percent of the total heap size.
Max used memory at stop	The maximum part of the heap used during the stop phase in bytes and percent of the total heap size.
Memory quota at download	The part of the total heap size available when program changes are sent to the controller. If the memory quota is exceeded an error icon is displayed.
Alarm Event	A summary of alarm and event information

In the System Diagnostics function block, “Memory size” is the total physical memory, minus executing firmware. This is sometimes also called the “heap”.

Memory usage is also displayed in the dialog “Heap Utilization” which can be displayed for each controller. The available memory is called “Non-Used Heap” and the rest is called “Used Shared Heap”.

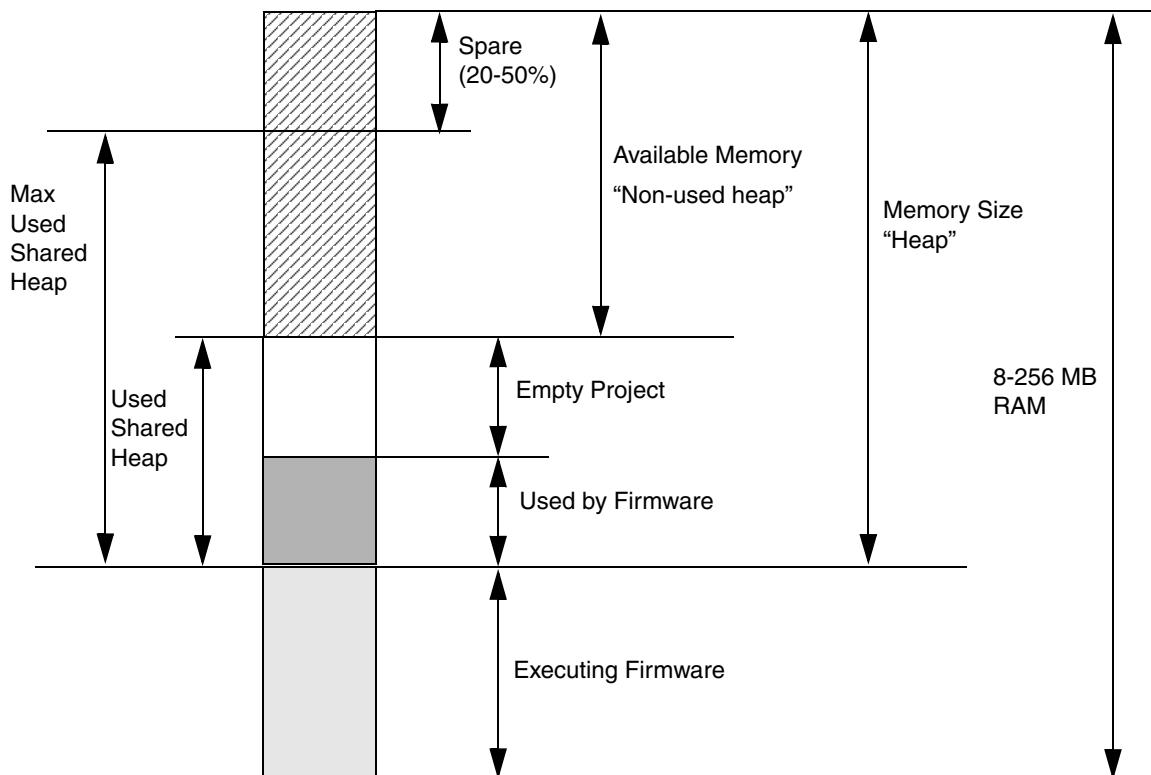


Figure 209. Memory organization

SystemDiagnosticsSM Function Block

This function block displays the RAM usage on the SM810/SM811/SM812 module. It also displays execution cyclicality and time for the SM diagnostic running in the PM module. Update is made by an operator request or cyclically if decided in the operator interface. Extreme values are locked and displayed. This locked information may be reset by a button in the operator interface. The type of SM is noted as well as if the SM does not exist.

The object is non-SIL. The object can not be executed in the time critical task.

SystemDiagnosticsSM Interaction Window.

The interaction window contains information about the supervised system, the loads on the System Memory, SM type, cyclic update rate, and startup status information.



The values displayed do not have any physical reality.

Trouble-Shooting Error Symptoms

Below are some examples of error symptoms and suggested measures.

Table 47. Examples of error symptoms and suggested measures.

Error Symptom	Measure
Control Builder fails.	<ol style="list-style-type: none"> 1. Click OK. 2. Copy the two crash dump files (see Crash Dumps for Analysis and Fault-Localization on page 467), the Start Log and the Heap Statistics Log files (if there are any). 3. Read the Session Log, and see if there is any information that indicates the source of the problem. 4. Try to start Control Builder. If it starts, select Help>About Control Builder M>List all information in the Project Explorer and the Control Builder System Information Report will be created. 5. Try to reproduce the fault, if possible. If the problem is reproducible, export the project with all dependencies and include the .afw file in the error report. 6. Check basic things, such as if the hard disk full. 7. If the fault appears during Offline to Online transfer, and it is possible to reproduce the fault, check the message written in the message pane, just prior to fault occurrence. This will give a hint about what operation (for example, sorting, compiling) and what application is involved in the problem. 8. Make an error report and include the log files.
A <i>Memory Full</i> message appears. The Heap Statistics log (SoftController) states that the heap is full.	Increase the heap size in SoftController, see Heap Statistics Log on page 458. Open Help > About and check the amount of free memory. Free memory should not be lower than 30%.

Table 47. Examples of error symptoms and suggested measures. (Continued)

Error Symptom	Measure
<p>A <i>Too many instances in application</i> message appears. The maximum number of above 65534 instances has been reached.</p>	<p>1. Try to reduce your application, see Statistics for Application on page 478.</p>
<p>Cannot create/open a control project. An <i>Action denied</i> message appears. License Error. When the Control Builder connects to an 800xA system a Control Builder license is checked out. If the license does not exist or the license count has been exceeded, a dialog window appears on the screen, displaying that the action was denied.</p>	<p>1. Close a running Control Builder client temporarily, in order to release a license. 2. Contact your System Administrator to extend the number of license features.</p>
<p>The MMS Server, OPC Server, Tool Routing Service, or SoftController fails.</p>	<p>1. Click OK. 2. Locate the two crash dump files (see Crash Dumps for Analysis and Fault-Localization on page 467). 3. Read the Session Log, and see if there is any information that points to the source of the problem. 4. Make an error report and include the log file.</p>

Table 47. Examples of error symptoms and suggested measures. (Continued)

Error Symptom	Measure
<p>The controller fails. The red F(ault) LED is On or flashing and the green R(un) LED is Off. See also Figure 95 on page 220.</p>	<ol style="list-style-type: none"> 1. Save the Post Mortem Memory Image on a Backup Media card inserted in the controller. See Dump of Post Mortem Memory Image on page 219. 2. Press the Init push-button on the controller until the Run LED starts to blink. Note that the controller will be empty if the red F LED is lit, that is, the application program has been deleted. 3. Fetch the Controller System log and save it, see Remote Systems Information on page 468. 4. Study the log, and find the marked reason for the stop (normally, at the end of the log). 5. Reload the application. 6. If an OPC Server for AC 800M is involved in communication, check the OPC Server function. 7. If possible, try to reproduce the problem. If the problem is reproducible, export the project with all dependencies and include the .afw file in the error report. 8. Make an error report and include the saved log files. See Error Reports on page 493. <p>Note that behavior similar to the example above is when there is no firmware installed in the controller (for example, when a new controller has been installed).</p>

Common Reason for Shut-Down AC 800M HI Controller

A number of different safety measures are used for supervision which all is potential reasons for a deliberate shutdown of the system. Some of the more common reasons are listed below:

Table 48. Common reasons for shutdown

Measure	Initial solution for download	Problem solution
FDRT	<p>Error text in log file</p> <p>FDRT tests not finished</p> <p>Increase the <i>FDRT</i> setting for the controller to 10000 ms.</p>	<p><i>FDRT</i> needs to be larger than the sum of the first scan execution time for all tasks in the largest application executing in the controller.</p> <p>The first scan execution time of a task is presented as <i>Max execution time</i> in the <i>Task Properties</i> dialog.</p>
Task latency	<p>Error text in CI log file</p> <p>Latency in task</p> <p>Increase the <i>Accepted latency</i> setting for each task (default 10%, minimum 10ms, max. 100%).</p>	<ul style="list-style-type: none"> • Run each task to get an idea of execution time. Configure the controller so that tasks are not scheduled to run simultaneously. Use the <i>Offset</i> setting for tasks to prevent them interfering with each other. See Overrun and Latency on page 178. • Let tasks with low accepted latency have higher priority • Use different priority for all tasks if possible.

Table 48. Common reasons for shutdown

Measure	Initial solution for download	Problem solution
Modulebus Scan Time	<p>Error text in log file</p> <p>Modulebus: Scan time error</p> <p>Increase the <i>modulebus scan time</i>.</p> <p>The default setting is 100 ms with also is the maximum value.</p>	<p>Calculate the minimum scan time possible for the I/O configuration that is used, using the formula in section Modulebus Scanning of Digital/Analog modules in the <i>System 800xA System Guide Technical Data and Configuration (3BSE041434*)</i>.</p> <p>Set the <i>modulebus scan time</i> to a value as high as possible but higher than the calculated minimum value and lower than the interval time of the fastest task using I/O signals.</p> <p>A Supervision function implemented in the controller generates a system alarm if the time taken to scan all modules exceeds the configured value +10 ms. If the configured value is set to 0, then the Supervision is disabled.</p>

Table 48. Common reasons for shutdown

Measure	Initial solution for download	Problem solution
Collection of Cold Retain Values CoRV (only SIL3)	<p>Cold Retain values are stored cyclically for SIL3 applications. If storing fails, it leads to a failed warm restart after power fail.</p> <p>Error text in log file:</p> <p>a) Failure - CoRV saving in PM</p> <p>or</p> <p>b) Time-out during collection of CoRV data in SM</p>	<p>The collection of Cold Retain values has failed for a SIL3 application.</p> <p>a) Select Tools > Maintenance > Analysis > Write Variable Memory to find the next error message starting the variable identified as the root of the problem e.g. CRC error detected. POUInstance index: 2, Varoffset: 127</p> <p>Analyze why the variable is detected as different between PM and SM, e.g. to check if the variable is an output parameter from an NonSIL/SIL restricted function.</p> <p>b) The controller load is too high to finish the collection of Cold Retain values in time. If persistent, this will lead to a failing warm restart at next power fail.</p>
Cold Retain Values missing for SIL3 application at power fail restart	<p>Controller shut down at power fail warm restart with error text in log file:</p> <p>Invalid CoRV data detected for some SIL-3 application</p>	<p>Restart controller and be aware of system events indicating problems with Cold Retain storage for SIL3 applications. Correct any problems as described above.</p>

Table 48. Common reasons for shutdown

Measure	Initial solution for download	Problem solution
Modulebus Discrepancy	<p>Modulebus frame discrepancy</p> <p>Example of text in log file:</p> <pre>4180 MBM1 SM vs PM CRC32 fail ad 0x44 (1,3,ES) E: Modulebus frame discrepancy Address AccessType Data(bin) Unused To Fr Mi Ma Ci PM: 404 WriteDigital 0100010101010101 00000000 44 00 00 0A 10 SM: 404 WriteDigital 0101010101010101 00000000 44 00 00 0A 10</pre>	<p>The rows beginning with PM: and SM: shows the data that has shown a discrepancy.</p> <p>Find the affected IO unit address and compare the bits to find out what channel that has a discrepancy. After the channel is located; analyze the application code used to manoeuvre the signal to find out if any nonSIL/SIL Restricted functions is used which leads to this discrepancy</p>



The Error printouts referred to in [Table 48](#), will only be presented in the CI log file.

For more information, see [Error Handler Log Entries](#) on page 430.

Restart after a controller shutdown



Always restart the PM with the flashing F(ault) LED by pressing the INIT push button. The backup PM will then automatically be restarted as well.

After a shutdown of a redundant Controller it's important to restart the PM that was running as primary at the time of the shutdown in order to get valid information. This PM is identified by a flashing F(ault) LED. Note that the F(ault) LED can temporarily be On (steady) while a Post Mortem Memory Image is written to the Backup Media card or flash rapidly (10 Hz) to indicate that a Memory Image is waiting to be written to a Backup Media card, as soon as a card is inserted in the PM, see [Figure 95](#) on page 220 for more details.

Controller Log

```
E 2004-12-08 15:01:24.539 ErrorHandler SM: Latency in task
with parameters: 1000 11 (1,4,ERS)
```

This PM has been intentionally stopped.

Reason:

- CPU stall timer has expired (Acknowledged)
- Manual shutdown was requested

Press init button or remove power to restart...

CI log

```
I 2004-12-08 15:01:24.248 (EHTask) [ERRORHANDLER] Send
ErrorReport to PM. ('Latency in task with parameters:', Sev =
4, ActionsToTake = 0xf)
```

Connection to Aspect Server

When the connection to the Aspect Server is broken, the Control Builder does not automatically indicate the loss of connection. However, if the user runs any action that requires the Aspect Server to be accessed (for example, saving or refreshing a type or program), a message is displayed.

If the connection to the Aspect Server is broken while some configuration is being saved in Control Builder, the Control Builder might stop functioning. The solution is to re-establish the connection to the Aspect Server, or to stop the Control Builder process using Task Manager.

Error Reports

An error report contains information to the problem in question. A detailed report is particularly valuable if your supplier's service department is to be involved.

The following information should *always* be included in an error report.

- Name of the person reporting the error (and the project, site, customer, etc.).
- Product (including the type of product and version).
- A listing of all information from the faulty system, such as the appropriate logs and reports, see [Dump of Post Mortem Memory Image](#) on page 219 and [Log Files](#) on page 447. The latter includes a great deal of information such as software version and revision, setup, etc. If the fault occurred during, or just after downloading a new version of the application program, the Control Builder Start Log and the Control Builder Session Log from the engineering station that performed the download should be included. Whenever a problem involving I/O handling occurs, it is very important to include a complete description of the I/O configuration.
- A description of the problem. Add all information that could help solve the problem, for example, what happened just before the error occurred, and other important circumstances. If it is possible to reproduce the error, describe the circumstances under which the error occurs. Sometimes it is advisable to create a small application to demonstrate the error, and add it to the error report.



If several systems are involved, information about the system configuration must be included (hardware type, etc.).

Appendix A Array, Queue and Conversion Examples

In this section you will find examples on how to handle arrays, queues, and some examples on how to use bit conversion functions.

Arrays

It is possible to create a one-dimensional array with elements of any type, that is, the elements can be a struct with variables of any type, or a single variable of any type. Using `PutArray` and/or `CopyArray`, it is possible to build a tree structure of arrays. Array elements are accessed direct via an index. A lower and upper boundary of the index should be defined. The array must first be created using `CreateArray`.

The size of an array is limited to 65,524 components (variables of simple data type).

Example

In this example, there is a data type *trec1* with the components *b* (bool), *i* (dint), and *st* (string).

The following variables are also needed:

Name	Data Type	Initial Value
MyArray	ArrayObject	
lrec	trec1	
lrec1	trec1	
lrec2	trec1	
lrec3	trec1	
Status	dint	
FirstScan	bool	TRUE

Create and initialize an array with 20 array elements of the type *trecl*.

Use an IF – THEN statement for the CreateArray function and let it be controlled by a variable, which is executed once during startup.

```
IF FirstScan THEN
FirstScan := false;
CreateArray(MyArray,1,20,lrec,status);
end_if;
```

Set up values for the different variables:

```
lrec1.b := TRUE
lrec1.i := 123
lrec1.st := A variable contaning the string 'Hello'
lrec2.b := FALSE
lrec2.i := 27
lrec2.st := A variable contaning the string 'BYE'
lrec3.b := TRUE
lrec3.i := 53
lrec3.st := A variable contaning the string 'BYE'
```

Set up the array contents:

```
PutArray (MyArray,1,lrec1,status);
PutArray (MyArray,2,lrec2,status);
PutArray (MyArray,3,lrec3,status);
```


The array now contains the following:

1	b = TRUE i = 123 st = 'Hello'
2	b = FALSE i = 27 st = 'BYE '
3	b = TRUE i = 53 st = 'BYE '
4	b = Undef. i = Undef. st = Undef.
20	b = Undef. i = Undef. st = Undef.

SearchStructComponent

SearchStructComponent is a boolean function which searches for a specific part in a record component. The corresponding components in *Struct* are scanned to find a part in the component which matches the *SearchComponent*.

```
Variable = SearchStructComponent(Struct, SearchIndex,
SearchCount, SearchStruct, SearchComponent, FoundStruct,
Status)
```

Table 49. *SearchStructComponent*

Parameter	Data type	Direction
Struct	AnyType	in_out
SearchIndex	dint	in_out
SearchCount	dint	in
SearchStruct	AnyType	in_out
SearchComponent	AnyType	in_out
FoundStruct	AnyType	in_out
Status	dint	in_out

The data type *SearchComponent* is either a single variable or a record containing a couple of variables corresponding to a subset of the record component in *Struct*. The *SearchComponent* could be either a boolean, integer, real or string data type or a sub record which contains these data types. The *SearchRecord* shall consist of a variable of *SearchType* and variables of the data types as the remaining variables in the record component and at the same positions.

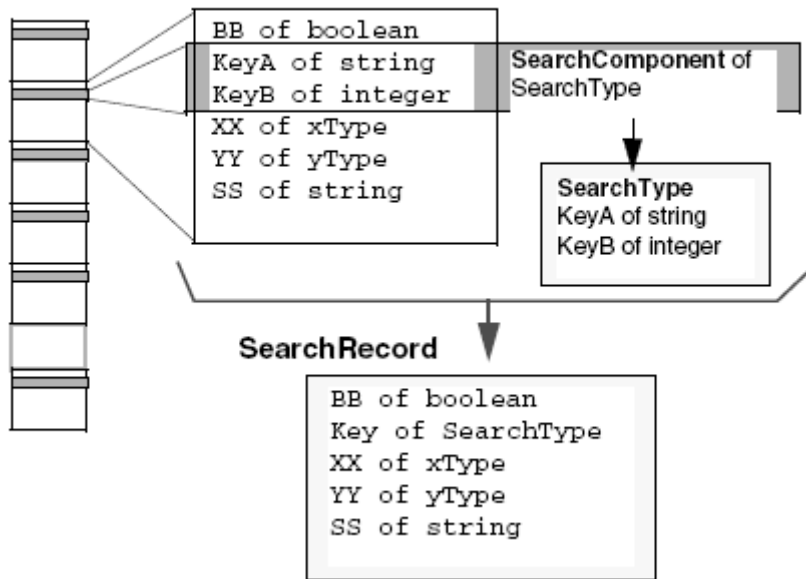


Figure 210. An example of the *SearchComponent* and a *SearchRecord*.

The *SearchComponent* may contain structured data types but the match is only carried out on the boolean, integer, real and string data types. The variables in *SearchComponent* of string data types must have the same length and content for a match. The content of string is not case sensitive and the space characters are treated as any other character. On match the whole record component is copied to *FoundStruct* and the function returns true.

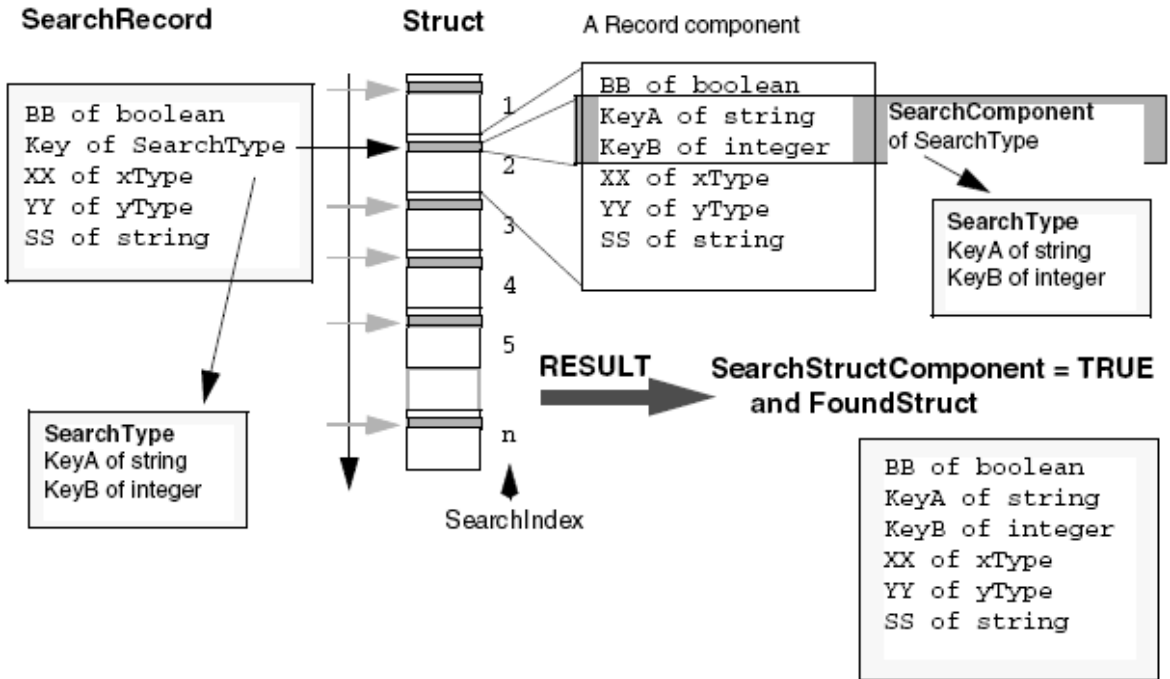


Figure 211. The working principal of the SearchStructComponent.

The search starts in the index *SearchIndex + 1* and ends at the first equivalent component located or, if there are no more sub-records, in the last component of the record.

A maximum number of record components given by *SearchCount* are scanned. The component, in which a match occurs, is returned in *FoundStruct* and the index is returned in *SearchIndex*.

Note that *SearchIndex* always points to the last record component that was scanned, even if no matching occurs. This index can then be used in a repeated call to find all occurrences of *SearchComponent* within the record.

Restrictions

The following data types in *Struct* will NOT be copied: *QueueObject*, *ArrayObject* and *tObject*.

The status returns:

- (1 Success)
 - The Search was successful
- (- 5 ErrTypeMismatch)
 - 1: Found sub-record was not of the same type as the *structRecord*.
 - 2: *SearchComponent* was not a subset of *SearchRecord*
- (- 6 ErrSizeMismatch)
 - 1: *SearchRecord* was not of the same size as the *StructRecord*.
 - 2: *SearchComponent* size is zero.
- (-30 ErrInvalidPar)
 - 1: *SearchIndex* was less than 0 or greater than the number of the *Struct* minus one.
 - 2: *SearchCount* was less or equal to zero.
 - 3: *SearchComponent* has no valid components (i.e., boolean, real, integer or string)

InsertArray

InsertArray (Array, Index, ArrayElement, Status)

Procedure: Inserts a new element in an array. All successive elements are moved one step, and the last element overwritten. Inserts the contents of *ArrayElement* into the record at position *Index* in the array *Array*. The records at position *Index + 1* to position *LastIndex* will be moved one position higher. The contents (even objects) of the record at position *LastIndex* will be lost. Variables of the data type *tObjects* will not be copied, unless the variable is an *ArrayObject*, then this array and its whole tree structure of arrays will be copied into an identical tree structure. If the record at position *Index* lacks some array in the tree structure, the array will be created.

Table 50.

Parameter	Data type	Direction
Array	ArrayObject	in_out
Index	dint	in
ArrayElement	AnyType	in_out
Status	dint	in_out

SearchArray

SearchArray(Array, SearchIndex, SearchCount, SearchElement, SearchComponent, FoundElement, Status)

This boolean function searches the array *Array* for a certain component in an array element. All elements in the array are scanned to find an element with a component (e.g. a string, or an entire record) that matches the search variable component.

The component *SearchComponent* in the element *SearchElement* is tested for equality with corresponding components in each array element. The function returns true if there is a find.

The search starts in the index *SearchIndex* + 1 and ends at the first equivalent component located or if there are no more elements in the array to be scanned. A maximum of number of array elements indicated by *SearchCount* are scanned. The array element, in which a find occurs, is returned in *FoundElement* and the index for the find is also returned in *SearchIndex*.

Note that *SearchIndex* always points to the last element that was scanned, even if no find occurs. This index can then be used in a repeated call in order to find all occurrences of *SearchComponent* within the array.

An error status is returned if:

- the index *SearchIndex* points outside array limits.
- the counter *SearchCount* is less than or equal to 0.
- the element *SearchElement* is not of the same type as *FoundElement*.
- the element *SearchElement* has a different size than *FoundElement*.
- the *SearchComponent* is not a part of the element *SearchElement*.

Table 51.

Parameter	Data type	Direction
Array	ArrayObject	in_out
SearchIndex	dint	in_out
SearchCount	dint	in
SearchElement	AnyType	in_out
SearchComponent	AnyType	in_out
FoundElement	AnyType	in_out
Status	dint	in_out

Example

Table 52. Data Type Definitions

Name	Data Type
trec1	Struct
b	Boolean
i	dint
s	String
tSearchStruct	STRUCT
b	Boolean
SSR	tSearchSubRec
tSearchSubStruct	Struct
i	dint
s	String

Table 53. Variables

Name	Data type	Initial value
Array	ArrayObject	
HitBoolean	Boolean	
HitRec	trec1	
Lrec	trec1	
lrec1	trec1	
lrec2	trec1	
lrec3	trec1	
Status	dint	
SearchRec	tSearchStruct	
FirstScan	Boolean	TRUE

Create and initialize an array with 20 array elements of type trec1.



The Create function may be in a Start_Code and in that case it is not necessary to use the IF -THEN statement and Firstscan variable.

```
IF Firstscan THEN
Firstscan = false;
CreateArray(Array,1,20,lrec,status);
ENDIF;
```

Set up values for the different variables e.g. via interaction objects:

```
lrec1.b <- TRUE
lrec1.i <- 123
lrec1.s <- "hello"
lrec2.b <- FALSE
lrec2.i <- 27
lrec2.s <- "BYE"
lrec3.b <- TRUE
lrec3.i <- 53
lrec3.s <- "BYE"
```


Set up array contents:

```
PutArray (Array,1,lrec1,status);
PutArray (Array,2,lrec2,status);
PutArray (Array,3,lrec3,status);
```

The array now contains the following:

Index	Contents
1	TRUE 123 'hello'
2	FALSE 27 'BYE '
3	TRUE 53 'BYE '
4	undefined
20	undefined

Figure 212. An example of an Array.

Access the array by index:

```
Index = 3;
GetArray (Array, Index, lrec, status);
```

lrec now contains:

```
TRUE 53 "BYE "
```

Now access the array by searching. First set up the search component.

```
SearchRec.SSR.i = 27;
```

SearchRec.SSR.s has its default value "BYE" Search a maximum of 10 array elements for the search component. A find occurs where the integer element is 27 and the string element is "BYE", in this case at array index no 2. Start searching in the first element number 1.

```
Index = 0;
IF SearchArray(Array, Index, 10, SearchRec, SearchRec.SSR,
HitRec, Status) THEN
IF Status > 0 THEN
HitBoolean = HitRec.b; (Save Boolean content of hit element)
ENDIF;
ENDIF;
```

Queues

A queue may consist of elements of any type, that is, the elements could be a struct with variables of any type, or a single variable of any type. Queue elements can be accessed at both ends of the queue, that is, only the first and last element can be accessed, but any element in the queue can be read. When using PutFirstQueue and GetFirstQueue, the queue act as a stack. When using PutLastQueue and GetFirstQueue, the queue will act as a FIFO queue. The size of the queue is not dynamic, and has to be defined. The number of elements in the queue is dynamic.

The size of a queue is limited to 65,524 components (variables of simple data type).

Example 1

The following structured variable *Item* is needed:

Name	Data Type	Initial Value
b	bool	TRUE
i	dint	123
st	string	'Hello'

The following variables are needed:

Name	Data Type	Initial Value
data1	Item	
data2	Item	
Queue	QueueObject	
Status	dint	
FirstScan	bool	TRUE
flag1	bool	
flag2	bool	

Create and initialize an array with 10 elements of data type item:

In an IF – THEN statement the *CreateQueue* function may be controlled by a first scan variable.

```

if FirstScan then
  FirstScan := false;
  CreateQueue( Queue := Queue,
    Size := 10,
    QueueElement := data1,
    Status := status );
end_if;
if flag1 then
  PutLastQueue( Queue := Queue,
    QueueElement := data2,
    Status := status );
  flag1 := false;
elsif flag2 then
  GetFirstQueue( Queue := Queue,
    QueueElement := data2,
    Status := status );
  flag2 := false;
end_if;

```

Example 2

The following parameters are needed:

Name	Data Type	Description
Size	dint	Max no. of elements in queue
InData	AnyType	In element, of same type as OutData
OutData	AnyType	Out element, of same type as InData
Put	bool	Put InData in queue on up edge
Get	bool	Get OutData from queue on up edge
Clear	bool	Clear contents of queue
Error	bool	Out: type or size of error

The following variables are needed:

Name	Data Type	Description
Queue	QueueObject	Queue object
PutState	bool state	
GetState	bool state	
Status	dint	

Code block 1 called Start_name

```
(*CreateQueue*)
CreateQueue (Queue, Size, InData, status);
Error := status < 0;
```

Code block 2 (queue statement)

```
PutState := Put;
GetState := Get;
if PutState:NEW and not PutState:OLD then
  PutLastQueue (Queue, InData, status);
  Error := status < 0;
end_if;
if GetState:NEW and not GetState:OLD then
  GetFirstQueue (Queue, OutData, status);
  Error := status < 0;
end_if;
if Clear then
  ClearQueue (Queue, status);
  Error := false;
end_if;
```

Conversion Functions

DIntToBCD

The DIntToBCD function converts an integer to a BCD value. An error status is returned if overflow occurs and no BCD value is produced.

Example

The following variables are needed:

Name	Data Type
N	dint
BCD	dint
Status	dint

Convert an integer into a BCD value:

N = 12345 (N is 0 0 0 1 2 3 4 5)

N can be divided into eight four-bit nibbles, where each nibble represents one BCD digit. The least significant nibble is 5, the next 4, etc. These nibbles can be written in binary form as below:

All four-bit nibbles	0000	0000	0000	0001	0010	0011	0100	0101				
which is equiv. to	00	000	000	000	000	010	010	001	101	000	101	
BCD as decimal value	0	0	0	0	0	0	7	4	5	6	5	

```
DIntToBCD ( N, BCD, Status ) ;
```

BCD now contains the value 74565.

BCDToDInt

BCDToDInt converts a BCD value to an integer. An error status is returned if the BCD value is illegal (no integer value in these cases).

Example

The following variables are needed:

Name	Data Type
N	dint
BCD	dint
Status	dint

Convert the BCD value into an integer:

BCD = 74565

BCD as decimal value	0	0	0	0	0	0	7	4	5	6	5
BCD as 32-bit pattern	00	000	000	000	000	010	010	001	101	000	101
BCD as four-bit nibbles				0000	0000	0000	0001	0010	0011	0100	0101

Each nibble represents one BCD digit. The least significant nibble is 5, the next 4, etc. These nibbles can be written in decimal form as: 0 0 0 1 2 3 4 5.

```
BCDToDInt ( BCD, N, Status ) ;
```

N now contains the value 12345.

ASCII

ASCII character codes

ASCII (American Standards Committee for Information Interchange) originally defined a set of codes for 128 characters and commands. Manufacturers later extended the ASCII codes to provide another 128 characters.

ASCII is a method of coding characters and command sequences, which is extensively used by manufacturers of peripheral equipment. Many devices transmit information in ASCII code (for example bar-code readers, keyboards) and many devices accept information in this form (for example VDUs and printers).

ASCII-coded strings allow for the transmission of non-printable characters and control characters. ASCII character sequences can be used to change the mode of a VDU display, or the character set of a printer.

Control Builder provides three procedures and one function manipulating ASCII strings (ISO Latin-1 only). These are useful when a device requires ASCII-coded information, and can be used to send ASCII-coded strings to printers, terminals etc.

Any ASCII character code may be used, thus it is possible to send control characters and sequences to switch printers and VDUs into various display modes. (Bold, Double Space, Reverse video etc.).

Before describing the procedures and functions available for ASCII strings, it is useful to examine the way in which an integer is stored in the system memory.

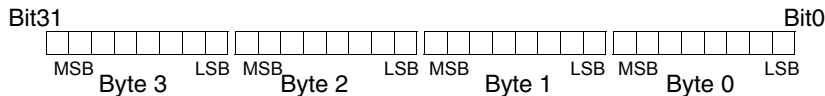


Figure 213. Integers are stored as four bytes in memory.

Integers are represented by a four-byte (32-bit) storage area. In normal usage, the bits are used to store both the value and the sign of the integer. This 4-byte storage space may also be used to store a series of values which represent an ASCII string.

Each ASCII character requires 1 byte of storage space. Therefore, it is possible to store up to 4 ASCII characters in a single memory area reserved for an integer.

The procedures below allow 1, 2 or 4 characters to be stored per integer.

Each ASCII character is coded with an integer value (in binary) between 0 and 255 (decimal). ASCII codes are normally represented as either their decimal equivalent, or as a hexadecimal number. If the character is represented as a hexadecimal number, then 2 digits are required for each character.

The hexadecimal digits, their decimal, and binary bit pattern equivalents are given in the table below:

Table 54. ASCII code representatives

Hexadecimal digit	Decimal digit	Binary bit pattern
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

The letter capital “A” is represented by the ASCII code 65_{10} or 41_{HEX} . Thus the letter “A” is stored as a byte having the bit pattern 0100 0001.

ASCII Conversion

StringToASCIIStruct (String1, NoOfCharsPerDint, DintStruct, Status)

This procedure converts a string to an ASCIIStruct. An ASCIIStruct consists of any number of integer components (see below).

The value of the parameter *NoOfCharsPerDint* determines how many ASCII characters are stored within each ASCII record component. This value can be 1, 2, 4 or -1, -2, -4 only. A negative value means that the sequence of bytes is reversed.

NoOfCharsPerDint determines how many character codes are packed into the four bytes available for the integer. If one character is stored per integer, then only the first eight least significant bits of each integer are used for storage, if positive, or the last eight, if negative.

DintStruct must be defined as follows: the type definition and its components can be given any name, but the components must all be of integer data type. The number of components (of integer type) should be decided based on the length of the string to be converted, and also the number of characters which are to be stored in each integer. The converted string may need to be transmitted to a peripheral device, so the characteristics of this device should also be taken into account.

The maximum length for any string is 140 characters, and if this maximum is to be stored in the minimum number of integer components, then this will require 35 integer components in the integer record (at four ASCII characters per integer). If you anticipate the need to store this number of characters, then an integer record of 35 integer components should be defined.

Status returns an indication of the result of the operation.

Storage with Different Character Packing Factors

When *NoOfCharsPerDint* is set to 1, each integer variable holds the value for one ASCII character. Thus the character capital “A” is stored as decimal 65 in the integer, as a bit pattern of 0100 (Nibble1) and 0001 (Nibble0).

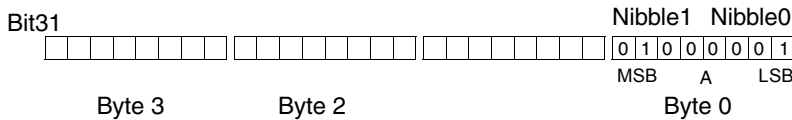


Figure 214. The ASCII code for “A” stored in an integer (packing = 1 character per integer)

When *NoOfCharsPerDint* is set to 2, each integer variable stores the value for two ASCII characters. The characters “AB” are stored as decimals 65 and 66 in the integer. The value 65 for “A” is stored in the first byte of the integer, and that for “B” in the second byte.

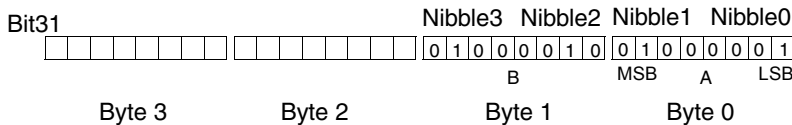


Figure 215. The ASCII codes for “AB” stored in an integer (packing = 2 characters per integer)

When *NoOfCharsPerDint* is set to 4, each integer variable contains the value for four ASCII characters. The characters “ABCD” are stored as decimals 65, 66, 67 and 68 in the integer. The value 65 for “A” is stored in the first byte of the integer, “B” in the second byte, “C” in the third byte, and “D” in the fourth byte.



Figure 216. The ASCII codes for “ABCD” stored in an integer (packing = 4 characters per integer)

Definition of DintStruct type

The appropriate length of an integer struct to store ASCII code is defined by the number of components required as follows.

Suppose we want to be able to store the maximum string length at a packing factor of 4 characters per integer. A data type called, for instance, *ASCIIMaxStringType*, should be defined consisting of 35 components which must be of integer data type called, for example *Chars1_4*, *Chars5_8* etc.

Usage

A string interaction is used to input the value of a string, (to a string variable called *String1*), which is to be converted to ASCII code. The code is stored in an integer struct called *IntStruct* which has 4 components (*Comp1* to *Comp4*).

The procedure call:

```
StringToASCIIStruct (String1, 1, IntStruct, Status1)
```

will write to the integer record components.

If the input string is “ABCD”, then the components will have the values 65, 66, 67 and 68, respectively. The literal value of 1 for the *NoOfCharsPerDint* determines that there is to be one character code in each component.

If *NoOfCharsPerDint* had been set to 2, then the first integer component would have the value 16961 (which is the decimal equivalent of 65 in the first byte and 66 in the second), and the second component would have the value 17475, which is the decimal equivalent of 67 in the first byte and 68 in the second. The other two bytes in each integer component are set to 0000.

Unused components

NoOfCharsPerDint determines how many bits are allocated for storage (8 bits – 1 byte per character) for a component. For example, if *NoOfCharsPerDint* is set to 2, then only the first two bytes are used in each component for data storage. The remaining bytes are set to 0 (zero).

This is illustrated below:

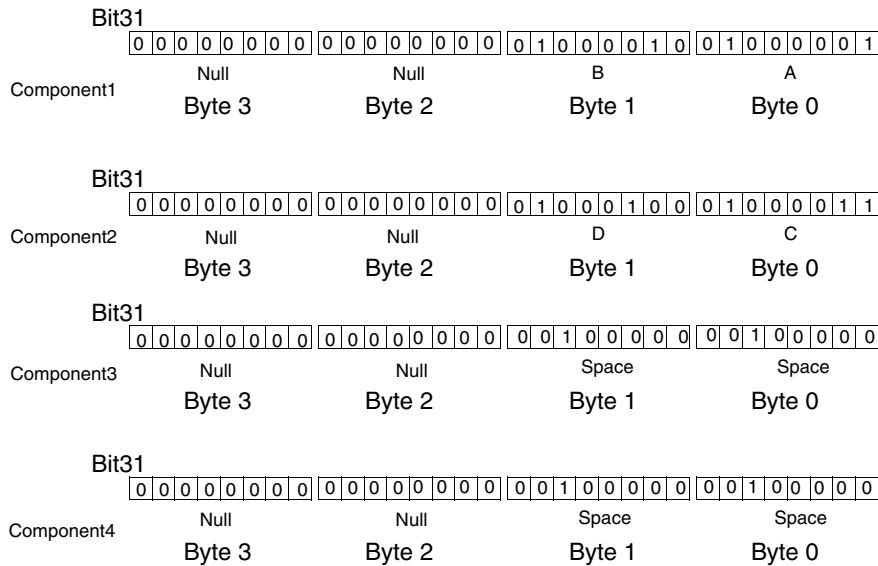


Figure 217. The diagram shows four integer components of an integer record. NoOfCharsPerDint has been set to 2, so that each component stores two ASCII characters. The character string “ABCD” has been transferred to the struct.

Note the following

If there are two characters per integer, the allocated storage areas Byte 0 and Byte 1 contain either the code for the string character, or if there is no character available, the code for a space (20_{HEX}). Unused bit positions (Bytes 2 and 3 in this case) contain zero.

Note:

- Characters from the string to be transferred are read from the current pointer position in the source string.
- Space characters are inserted into the allocated storage areas within each component. They are also inserted into all records to which no characters have been transferred, for example, if the actual string requires less than the number of components available for storage.
- An error status is returned to the value of *Status*, if the string to be transferred is longer than the storage space allocated. In this case, no transfer of any part of the string occurs.

ASCIIStructToString(DIntStruct, NoOfCharacters, NoOfCharsPerDInt, String1, Status)

This procedure is the reverse of *StringToASCIIStruct* described above. It takes an integer struct, which contains the codes for an ASCII string, and recreates the string from the values in the components of the record. (See *StringToASCIIStruct* for full details of the structure of the integer struct and the encoding method.)

The component values of the integer struct, *DIntStruct* are read and translated to the value of the destination string, *String1*.

The value of the parameter *NoOfCharacters* determines how many ASCII characters are read from the source record, *DIntStruct*, and the value of the parameter *NoOfCharsPerDInt* informs the procedure how many characters are to be expected in each integer component. *Status* returns an indication of the result of the operation.

The *DIntStruct* parameter must be structured as an integer struct, that is, it must have integer components only. (See details in *StringToASCIIStruct*.)

NoOfCharacters and *NoOfCharsPerDInt* may be variables, module parameters or literals.

Usage

Suppose the integer struct *DIntStruct* from the previous example is to be converted back to a string. The destination string is called *String1* and the three characters are to be copied. It is known that the original storage protocol defined 2 characters per integer component.

The following code will perform the task:

```
ASCIIStructToString(DIntStruct, 3, 2, String1, Status2)
```

After execution the value of *String1* value will be “ABC”.

Note

- The number of characters per integer of the original record must be known, only values of 1, 2, 4 or -1, -2, -4 are allowed.
- The new output string will be inserted at the current pointer position in the destination string.
- An error status is reported as a value to *Status* if the generated string results in a new string which is longer than the permitted length for the destination string.

Appendix B System Alarms and Events

This section is divided in sub-sections for system alarms and system simple events and it describes system alarms and system simple events from a controller perspective. Additional information can also be found in the Control Builder online help.

General

OPC Server

The OPC Server objects in Control Structure are created automatically when a Control Builder project is opened. These OPC Server objects have an associated Aspect Object as source object for the alarm/event in Plant Explorer, for the System Alarm&Events generated by the OPC Server.

System alarms and system simple events generated within OPC server can be divided in two general groups regarding to originating part of the OPC server (source).

- Software
- Subscriptions

Controller

System alarms and system simple events generated within controller can be divided in two general groups regarding to originating part of the controller (source).

- Software generated system alarms and system simple events.
- Hardware generated system alarms and system simple events.

OPC Server – Software

All system alarms and system simple events triggered by base code executing in OPC Server belong to this group. This group is further divided into appropriate parts uniquely identified by source name suffix.

- `_SWFirmware` – for common base code
- `_SWDataAccess` – for OPC Data Access specific code
- `_SWAlarmEvent` – for OPC Alarm and Event specific code

The SrcName shall be automatically formed as:

SrcName = SystemIP address- SrcNameSuffix

Example: SrcName = 172.16.85.90:200-`_SWFirmware`

SrcNameSuffix = `_SWFirmware`

System Alarm `HeapFull`

```
SrcNameSuffix = _SWFirmware;  
Condition name = HeapFull;  
Message = "(1000) The Heap is full";  
SeverityLevel = High;
```

SrcNameSuffix = `_SWDataAccess`

System Simple Event `SaveColdRetainFailed`

Generated when OPC Data Access server can not save cold retain files for an application.

```
SrcNameSuffix = _SWFirmware;  
Message = "(5000) Save Cold Retain failed for {1}";  
{1} = The name of the application.  
SeverityLevel = Medium;
```

SrcNameSuffix = _SWAlarmEventSystem Simple Event **AlarmNotUnique**

Generated when OPCAE server discover that there are two alarms with same combination SouceName ConditionName defined in two different controllers.

```
SrcNameSuffix = _SWAlarmEvent ;  
Message = "(6000) Alarm not unique {1}, {2}";  
{1} = Source name of the alarm  
{2} = Condition name of the alarm  
SeverityLevel = Medium;
```

System Simple Event **AlarmHandler overflow**

Generated when an item in the EventHandler must be deleted because of overflow. If there is space again in the EventHandler, the system initializes an AlarmSummary and updates the missing information. The size of the EventHandler is limited by the system variable MaxNoOfAlarms.

```
SrcNameSuffix = _SWAlarmEvent ;  
Message = "(6001) AlarmHandler overflow. MaxNoOfAlarms exceeded";  
SeverityLevel = Medium;
```

System Simple Event **FailedToSubscribe**

Generated when a try from OPC AE server to subscribe to a certain control system was not successful. The corresponding control system name shall be concatenated to this message.

```
SrcNameSuffix = _SWAlarmEvent;  
Message = "(6002) Failed to subscribe on {1}";  
{1} = The IP address of the control system.  
SeverityLevel = Medium;
```

System Simple Event **Overflow in queue to OPC client**

Generated after an overflow of the event queue to an OPC client queue and when the queue is filled less than 75% of the actual size. The system event is generated and sent to the client to announce the overflow. On overflow the latest event is thrown away. The size of every event queue to an OPC client queue is limited by the system setting "Queue size".

```
SrcNameSuffix = _SWAlarmEvent;  
Message = "(6003) Overflow in queue to OPC client";  
SeverityLevel = Medium;
```

OPC Server – Subscription

OPC server can subscribe a number of controllers from both Data Access and Alarm and Event part. Thus, each subscribed controller may have one or two system alarms for its disposal, depending on number of subscription to controller from OPC server. These system alarms must be created in a moment of corresponding connection i.e. subscription establishing.

The **SrcNameSuffix** for *Data Access* subscriptions group is:

```
SrcNameSuffix = SubDataAccess  
Example: SourceName = 172.16.85.90:22-SubDataAccess
```

The **SrcNameSuffix** for *Alarm and Event* subscriptions group is:

```
SrcNameSuffix = SubAlarmEvent  
Example: SourceName = 172.16.85.90:22-SubAlarmEvent
```

The **ConditionName** for these system alarms is supposed to provide a unique combination of SrcName and ConditionName (since SrcName is the same for whole category). Thus, ConditionName has form that contains controller IP address.

```
Example: ConditionName = 172.16.85.90:2-ConnectionError
```

The following category of system alarms and system simple events handle errors and warnings concerning connection towards subscribed controllers.

SrcNameSuffix = SubDataAccess

Each controller subscribed from Data Access should have one system alarm for its disposal. Note that these system alarms shall be:

- defined when a new subscription (connection) is established
- activated when an error occurs on this connection
- inactivated when all errors are corrected or disappeared
- deleted when subscription is removed

Condition name has form that includes subscribed controller IP address. It is created dynamically but last part is always the same: "-ConnectionError".

Example: Condition name = 10.46.37.121:2-ConnectionError.

System Alarm ConnectionError to DA subscription

SrcNameSuffix = SubDataAccess;

Condition name = -ConnectionError;

Message = "(5500) Connection error to DA subscribed controller";

Severity Level = Critical;

SrcNameSuffix = SubAlarmEvent

Each controller subscribed from Alarm and Event should have one system alarm for its disposal. Note that these system alarms shall be:

- defined when a new subscription (connection) is established
- activated when an error occurs on this connection
- inactivated when all errors are corrected or disappeared
- deleted when subscription is removed

Condition name has form that includes subscribed controller IP address. It is created dynamically but last part is always the same: "-ConnectionError".

Example: Condition name = "10.46.37.121:2-ConnectionError".

System Alarm ConnectionError to AE subscription

```
SrcNameSuffix = SubAlarmEvent;  
Condition name = -ConnectionError;  
Message = "(6500) Connection error to AE subscribed controller";  
Severity Level = Critical;
```

Controller – Software

All system alarms and system simple events triggered by base code belongs to this group.

This is important to note that system alarms and system simple events issued by protocol specific code may belong to this group. Normally system alarms and system simple events issued by protocol specific code are handled within 'Hardware group'. Under certain circumstances when it is necessary to define errors or warnings that are not covered by HW state error handling, this group i.e. corresponding dedicated SrcNameSuffix should be used. The following set of source name suffixes are defined for this group.

- `_SWFirmware` - for base code
- `_SW1131Task` - for 1131 task execution specific code
- `_SWTargets` - for HW and OS abstraction layer of the base code
- `_SWInsum-`, `_SWS100-`, `_SWMB300-`, `_SWProfibus-`, `_SWModbus-`¹for protocol specific code

SrcNameSuffix = _SWFirmware**System Alarm HeapFull**

```
SrcNameSuffix = _SWFirmware;  
Condition name = HeapFull;  
Message = "(1000) The Heap is full";  
SeverityLevel = High;
```

1. System alarms and system simple events generated by respective communication protocol are described in the online help function for respective protocol.

System Alarm ErrorHandler sum alarm

SrcNameSuffix = _SWFirmware
Condition name = ErrorHandler;
Message = "(1001) ErrorHandler sum alarm created";
SeverityLevel = Medium;

System Alarm Data transfer failed during FW-upgrade of Alarm&Event

This alarm is generated when Alarm&Event failed in the transfer of Alarm&Event data from Primary CPU to Trainee CPU. It shows how many items of different Alarm&Event data that failed. The consequence after upgrade could be that inactive alarms disappear but active alarms will be activated again.

SrcNameSuffix = _SWFirmware;
Condition name = HeapFull;
Message = "(1002) Alarm&Event failed in FW-upgrade. No of Static alarms = {1}. No of Simple events = {2}. No of Dynamic alarms = {3}. No of SOE-events = {4}";
{1} = Number of failed items.
{2} = Number of failed items.
{3} = Number of failed items.
{4} = Number of failed items.
SeverityLevel = High;

System Simple Event EventNotificationLost

An event notification was lost. This can happen when the particular OPC-Server or printer queue containing event notification is full. A system simple event is generated when there is space again in this queue. After this the missing information about alarms in the subscribing systems-OPC Servers is updated, but this does not mean that all missed events are regenerated.

SrcNameSuffix = _SWFirmware;
Message = "(1010) Lost event notification(s) to {1}";
{1} = The remote systems (the OPC Servers) IP address when generated event indicates full OPC-Server queue or with string "local printer" when there is a lost event notification from a filled buffer in printer queue.
Severity Level = Medium;

System Simple Event Alarm definition failed

An attempt to define a process alarm in controller, or a system alarm in controller or in OPC server was not successfully completed.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1011) Alarm definition failed for {1}, {2}";  
{1} = Source name  
{2} = Condition name  
Severity Level = Low;
```

System Simple Event Undeclared External event

A low level event issued by external device is received, but no declaration was found in applications.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1012) Undeclared external event; {1}";  
{1} = Signal ID and new value delivered by low level event.  
Severity Level = Medium;
```

System Simple Event No enable/disable of alarms in SIL applications

An attempt enable/disable an alarm (via MMS) in a SIL application which is not permitted.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1013) No enable/disable of alarms in SIL applications ({1}, {2})";
```

This message is concatenated with source name and condition name of the alarm.

```
Severity Level = Medium;
```

System Simple Event Event notification(s) lost during firmware upgrade

Generated if events are lost during firmware upgrade

```
SrcNameSuffix = _SWFirmware;  
Message = "(1014) Event notification(s) lost during firmware upgrade"  
SeverityLevel = Medium
```


System Simple Event Alarm definition(s) failed during firmware upgrade

Generated if there are attempting to create alarms during firmware upgrade.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1015) Alarm definition(s) failed during firmware upgrade"  
SeverityLevel = Medium
```

System Simple Event CommandedSwitchover

The system event below is issued when a commanded switchover has successfully been executed.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1020) CPU Switchover was commanded";  
SeverityLevel = Medium;
```

System Simple Event CommandedSwitchoverFailed

The system event below is issued when a commanded switchover has been unsuccessfully executed.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1021) CPU Switchover command failed";  
SeverityLevel = Medium;
```

System Simple Event Reset of backup CPU was commanded

The system event below is issued when a commanded reset of backup CPU has successfully been executed.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1022) Reset of backup CPU was commanded";  
SeverityLevel = Medium;
```

System Simple Event Reset of backup CPU command failed

The system event below is issued when a commanded reset of backup CPU has unsuccessfully been executed.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1023) Reset of backup CPU command failed";  
SeverityLevel = Medium;
```

System Simple Event Error found in DataToSimpleEvent

The system event below is generated during calls to DataToSimpleEvent function block.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1030) AE setting NamValItem/LogStrings to low";  
Message = "(1031) Error in FB parameters";  
Message = "(1032) Data overflow in communication buffer";  
SeverityLevel = Medium;
```

System Simple Event Reset of controller forces performed

System event generated from Access Management. Message when Override Control has made a reset of controller forces.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1033) Reset of controller forces performed";  
SeverityLevel = Medium;
```

System Simple Event Ack of event denied

System event generated from Access Management, when acknowledgement of an alarm is denied.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1034) Acknowledge of event denied ({1}, {2})";  
{1} = source name of the alarm  
{2} = condition name of the alarm  
SeverityLevel = Medium;
```

System Simple Event No configuration image found at compact flash card

The system event below is issued when a compact flash card, without a configuration image, is detected during startup of controller.

```
SrcNameSuffix = _SWFirmware;  
Message = ">(1040) No configuration image found at compact flash card";  
SeverityLevel = Medium;
```

System Simple Event Configuration image found at compact flash card is corrupt

The system event below is issued when a compact flash card, with a corrupt configuration image, is detected during startup of controller

```
SrcNameSuffix = _SWFirmware;  
Message = "(1041) Configuration image found at compact flash is  
corrupt";  
SeverityLevel = Medium;
```

System Simple Event Configuration image found at compact flash does not match controller

```
SrcNameSuffix = _SWFirmware;  
Message = "(1042) Configuration image found at compact flash does not  
match controller"  
SeverityLevel = Medium
```

System Simple Event is started from compact flash

```
SrcNameSuffix = _SWFirmware;  
Message = "(1043) is started from compact flash"  
SeverityLevel = Medium
```

System Simple Event Configuration image found at compact flash has different format

```
SrcNameSuffix = _SWFirmware;  
Message = "(1044) Configuration image found at compact flash has  
different format"  
SeverityLevel = Medium
```

System Simple Event Configuration image found at compact flash does not match controller

The system event below is issued when a compact flash card, with a configuration image created for another type of CPU, is detected during startup of controller.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1042) Configuration image found at compact flash does not  
match controller"  
SeverityLevel = Medium;
```

System Simple Event is started from compact flash

The system event below is issued when a compact flash card, with a valid configuration image, is detected during startup of controller.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1043) is started from compact flash"  
SeverityLevel = Medium;
```

System Simple Event Configuration image found at compact flash has not equal format

The system event below is issued when a compact flash card, with a configuration image created in a format not supported, is detected during startup of controller.

```
SrcNameSuffix = _SWFirmware;  
Message = "(1044) Configuration image found at compact flash has  
different format"  
SeverityLevel = Medium;
```

System Simple Event

```
SrcNameSuffix = _SWFirmware;  
Message = "(1045) Write attempt to constant variable {1} of instance  
{2}";  
Severity Level = High;
```

System Simple Event

SrcNameSuffix = _SWFirmware;

Message = "(1046) System variable LogConstAbuse set to 0 since limit on { 1 } messages reached";

Severity Level = High;

This message can occur on process alarms when the alarms have not executed yet. For example, after an OLU. It should only occur temporarily and will disappear when the execution has started and the alarm have changed state or when OPC/AE clients (example PPA) are refreshed.

Message = "(1047) The message text is temporarily unavailable since the alarm is issued before 1131 has been run.";

Severity Level = Medium;

SrcNameSuffix = _SW1131TaskSystem Alarm **TaskAbort**

SrcNameSuffix = _SW1131Task;
Condition name = TaskAbort;
Message = "(2000) Execution time too long in Task {1}";
{1} = Task name will be added to message, for example, "Execution time too long in Task Fast"
Severity Level = Fatal;

System Simple Event **Interval time in ordinary tasks inc**

SrcNameSuffix = _SW1131Task;
Message = "(2001) Interval time in ordinary tasks increased {1}%";
{1} = The increase of the interval time in percent with the precision of one decimal.
Severity Level = Medium;

System Simple Event **Interval time in ordinary tasks dec**

SrcNameSuffix = _SW1131Task;
Message = "(2002) Interval time in ordinary tasks decreased {1}%";
{1} = The decrease of the interval time in percent with the precision of one decimal.
Severity Level = Medium;

System Simple Event **Interval Time was changed**

Only used for tasks executing at Time-Critical priority.

SrcNameSuffix = _SW1131Task;
Message = "(2003) Interval time changed to {1} ms. Task={2}";
{1} = New interval time ,
{2} = Name of the task.
Severity Level = Medium;

System Alarm Latency high in normal tasks

The alarm is activated when actual latency is 70 % of max latency.

```
SrcNameSuffix = _SW1131Task;  
Message On = "(2004) Latency high in task {1}, {2} ms"  
{1} = Name of the task,  
{2} = Actual latency.  
Message Off = "(2004) Latency high inactive "  
Condition name = High Latency  
SeverityLevel = Medium
```

System Alarm Latency high in time critical task

The alarm is activated when actual latency is 70 % of max latency.

```
SrcNameSuffix = _SW1131Task;  
Message On = "(2005) Latency high in task {1}, {2} ms"  
Message Off = "(2005) Latency high inactive "  
{1} = Name of the task,  
{2} = Actual latency.  
  
Condition name = High Latency  
SeverityLevel = Medium
```

SrcNameSuffix = _SWTargets**System Simple Event RCU error detected in the Primary CPU**

```
SrcNameSuffix = _SWTargets;  
Message = "(4000) Primary CPU: RCUError(0x{2})";  
{2} = Content of the RCU Error Register in hexadecimal format.  
Severity Level = High;
```

This event is issued from the RCU Driver if redundancy has been shut down due to an internal error in the RCU Driver found at the end of synchronization.

System Simple Event

```
SrcNameSuffix = _SWTargets;  
Message = "(4025) Failed to set RCU Driver state to
```

```
eRCUTakeoverPossible";
Severity Level = High;
```

This event is issued from the RCU Driver if redundancy has been shut down due to an internal error in the RCU Driver found at the start of synchronization.

System Simple Event

```
SrcNameSuffix = _SWTargets;
Message = "(4026) Failed to set RCU Driver state to eRCUNormal";
Severity Level = High;
```

This event is issued from the RCU Driver if Online Upgrade has been suspended due to an internal error in the RCU Driver.

System Simple Event

```
SrcNameSuffix = _SWTargets;
Message = "(4027) Failed to set RCU Driver state to eRCUOLU";
Severity Level = High;
```

System Simple Event **RCU test error detected in the Primary CPU**

```
SrcNameSuffix = _SWTargets;
Message = "(4001) Primary CPU: RCUTestError({2}, 0x{3})";
{2} = Test Number
1 = RCU Register test
2 = Log Parity test
3 = Log test
4 = Log Range test
5 = I O Emulation test
6 = CPU Bus Timeout test
{3} = The Error status is printed in hexadecimal format.
Severity Level = High;
```

System Simple Event **Dual test error detected in the Primary CPU**

```
SrcNameSuffix = _SWTargets;
Message = "(4002) Primary CPU: DualTestError({2}, 0x{3})";
{2} = The Dual Test status (see Table 55)
```


{3} = The Error status is printed in hexadecimal format.
Severity Level = High;

Table 55. Dual Test status.

Message	Description
CPUCEXBusMsgSendError	Failed to send test message to the Backup CPU
CPUCEXBusMessageError	Failed to receive test message from the Backup CPU
CheckpointTestError	Failed to upgrade memory of the Backup CPU

System Simple Event **Backup CPU CEX-Bus test error detected in the Primary CPU**

```

SrcNameSuffix = _SWTargets;
Message = "(4003) Primary CPU: BkpCEXBusTestError({2}, 0x{3})";
{2} = The Test status (see Table 56)
{3} = The Error status is printed in hexadecimal format.
Severity Level = High;

```

Table 56. Test status from Backup CPU

Message	Description
CPUCEXBusMsgSendError	Failed to send test message to the Backup CPU
CPUCEXBusMessageError	Failed to receive response message from the Backup CPU
CEXBusTestError	Failed to test the CEX-Bus interface in the Backup CPU

System Simple Event **Error detected in the Primary CPU**

```

SrcNameSuffix = _SWTargets;
Message = "(4004) Primary CPU: {2} in state {3}";
{2} = The name of the detected error (see Table 57)
{3} = The state when the error was detected.
Severity Level = High;

```

Table 57. The name of the detected error.

Message	Description
SDCError	RCU Service data channel error
RCUConnectorOpen	The RCU Link cable is not connected to the own CPU
RCUOtherConnectorOpen	The RCU Link cable is not connected to the peer CPU
RCUDrvError	Failed when calling the RCU driver

Table 57. The name of the detected error.

Message	Description
InitCommError	Failed to initialize interrupt handling with the peer CPU
InformCommParamError	Failed to inform other CPU about communication parameters
GetCommParamError	Failed to get communication parameters from other CPU
BkpCPUNotAlive	The Backup CPU is not alive
BkpCPUCEXBusError	Backup CPU not connected to the CEX-bus
BkpCPUIllegalExternalState	Backup CPU has an illegal External state
Timeout	Backup CPU has not sent a response message within a specified timeout time
CloningStartError	Failed to start cloning in state Upgrading
CloningNotCompletedError	Cloning not completed in state Unconfirmed
CloningError	Cloning failed in state Synchronized
BkpFirmwareError	Backup CPU's firmware id not equal to Primary CPU's firmware id

System Simple Event **A Backup CPU is recognized and started**

```
SrcNameSuffix = _SWTargets;
Message = "(4005) Primary CPU: Backup CPU started";
Severity Level = Medium;
```

System Simple Event **The system has reached the Synchronized state**

The Backup CPU is ready to take-over if the Primary CPU fails

```
SrcNameSuffix = _SWTargets;
Message = "(4006) Primary CPU: Synchronized state";
Severity Level = Medium;
```

System Simple Event Switchover has occurred

SrcNameSuffix = _SWTargets;
 Message = "(4007) Switchover to {2} has occurred";
 {2} = "Lower CPU" or "Upper CPU"
 Severity Level = Medium;

System Simple Event Report of Backup CPU error after a switchover

SrcNameSuffix = _SWTargets;
 Message = "(4008) Primary CPU: {2} in {3}";
 {2} = The error reported from the backup CPU
 {3} = The position reported from the backup CPU
 Severity Level = Medium;

System Simple Event The Backup CPU has stopped

SrcNameSuffix = _SWTargets;
 Message = "(4009) Primary CPU: Backup CPU stopped ({2})";
 {2} = Stop reason (see [Table 58](#))
 Severity Level = High;

Table 58. Stop reason.

Message	Description
BkpCPUCEXBusError	Backup CPU not connected to the CEX bus
BkpHaltRequest	A Backup CPU problem has been detected in the Primary CPU. The Backup CPU however seems fully alive
BkpCPUNotAlive	The Backup CPU has stopped or been removed without reporting its status to the Primary CPU
Status sent from backup CPU	Backup CPU status received via the CEX bus

System Simple Event The Primary CPU has halted

SrcNameSuffix = _SWTargets;
 Message = "(4010) Primary CPU: CPU halted";
 Severity Level = High;

System Simple Event RCU error detected in the Backup CPU

SrcNameSuffix = _SWTargets;
 Message = "(4020) Backup CPU: RCUError(0x{2})";
 {2} = The contents of the RCU Error Register in hexadecimal format.
 Severity Level = High;

System Simple Event RCU test error detected in the Backup CPU

SrcNameSuffix = _SWTargets;
 Message = "(4021) Backup CPU: RCUtestError({2}, 0x{3})";
 {2} = Test Number (see [Table 59](#))
 {3} = Error Status. in hexadecimal format.
 Severity Level = High;

Table 59. Test Number

Test Number	Error Status
1	RCU Register test
2	Log Parity test
3	Log test
4	Log Range test
5	I/O Emulation test
6	CPU Bus Timeout test

System Simple Event Dual test error detected in the Backup CPU

SrcNameSuffix = _SWTargets;
 Message = "(4022) Backup CPU: DualTestError({2}, 0x{3})";
 {2} = Dual Test status (see [Table 60](#))
 {3} = Error Status. in hexadecimal format.
 Severity Level = High;

Table 60. Dual Test status.

Message	Description
CPUCEXBusMsgSendError	Failed to send test message to the Primary CPU
CPUCEXBusMessageError	Failed to receive test message from the Primary CPU
RCUDrvError	Failed when calling the RCU driver to set threshold value for the Log Data Buffer

System Simple Event **Error detected in the Backup CPU**

SrcNameSuffix = _SWTargets;
 Message = "(4023) Backup CPU: {2} in state {3}";
 {2} = The name of the detected error (see [Table 61](#))
 {3} = The state when the error was detected.
 Severity Level = High;

Table 61. The name of the detected error.

Message	Description
SDCError	RCU Service data channel error
RCUConnectorOpen	The RCU Link cable is not connected to the own CPU
RCUOtherConnectorOpen	The RCU Link cable is not connected to the peer CPU
RCUDrvError	Failed when calling the RCU driver
InitCommError	Failed to initialize interrupt handling with the peer CPU
InformCommParamError	Failed to inform other CPU about communication parameters
GetCommParamError	Failed to get communication parameters from other CPU

Table 61. The name of the detected error. (Continued)

Message	Description
EqualityCheckFailed	Memory upgrading of Backup CPU has failed
RCUMessageHaltReceived	A Halt request has been received from the Primary CPU
PrimCPUExitConnection	Primary CPU has exit connection

System Simple Event **The Backup CPU has halted**

```
SrcNameSuffix = _SWTargets;
Message = "(4024) Backup CPU: CPU halted";
Severity Level = High;
```

System Simple Event **Stopped due to ModuleBus inaccessible from Backup CPU**

This event is issued from the MBTestMC unit if the Backup CPU has been stopped due redundancy supporting modules on the module bus turned out to be inaccessible from the Backup CPU.

```
SrcNameSuffix = _SWTargets;
Message = "(4030) Stopped due to ModuleBus inaccessible from Backup CPU";
Severity Level = "High";
```

This event is issued from the RCU Driver if the Backup CPU has been halted due to an overload situation in the redundancy control HW.

System Simple Event

```
SrcNameSuffix = _SWTargets;
Message = "(4028) RCU LDB overflow has occurred in Backup/trainee PM";
Severity Level = High;
```

This event is issued if the Backup CPU has been halted during start-up due to that it uses the same MAC address as the Primary CPU. (This can happen if the original Primary CPU unit has been removed from a redundant controller and the same unit is later re-inserted as spare part in the same running controller.)

System Simple Event

SrcNameSuffix = _SWTargets;

Message = "(4042) Backup CPU has the same MAC Address as Primary CPU";

Severity Level = High;

System Simple Event **Switched over due to ModuleBus inaccessible from Primary CPU**

This event is issued from the MBTestMC unit if a switch-over occurred due to redundancy supporting modules on the module bus turned out to be inaccessible from the Primary CPU.

SrcNameSuffix = _SWTargets;

Message = "(4031) Switched over, ModuleBus inaccessible from Primary CPU";

Severity Level = High;

Events from Network Interface Supervision

System Simple Event **Backup CPU halted: Bad Network interface**

This event is issued from the NIS primary task if the Backup CPU has been halted due to both network interface in Backup CPU are not working properly.

SrcNameSuffix = _SWTargets;

Message = "(4040) Backup CPU halted: Bad Network interface";

Severity Level = High;

Events from Checking of Available MAC address in BackupSystem Simple Event **No MAC address in Backup CPU**

This event is issued to the primary PM if the backup PM has no MAC address.

```
SrcNameSuffix = _SWTargets;  
Message = "(4041) No MAC address in backup PM";  
Severity Level = High;
```

Events from Modulebus driverSystem Simple Event **Diverse pointer check**

This event is issued from the check of pointers to the DPM which is used in all accesses to read/write data to/from IO modules.

```
SrcNameSuffix = _SWTargets;  
Message = "(4050) Fatal Error in diverse pointer check";  
Severity Level = Fatal;
```

System Simple Event **Failed to send message to queue**

```
SrcNameSuffix = _SWTargets;  
Message = "(4051) Mbus msgQ failed: control of Primary/Backup Leds  
not run";  
Severity Level = Low;
```

System Simple Event **Null pointer**

```
SrcNameSuffix = _SWTargets;  
Message = "(4052) Null pointer check failed";  
Severity Level = Fatal;
```

System Simple Event **Failed to create message queue**

```
SrcNameSuffix = _SWTargets;  
Message = "(4053) Failed to create message queue";  
Severity Level = High;
```

System Simple Event **Test of RAM Error in MBM1 failed**

```
SrcNameSuffix = _SWTargets;  
Message = "(4054) Cyclic test of Ram Error in MBM1 failed";  
Severity Level = Critical;
```

System Simple Event **Runtime RAM Error in MBM1**

```
SrcNameSuffix = _SWTargets;  
Message = "(4055) Runtime Ram Error in MBM1";  
Severity Level = Critical;
```

System Simple Event **Diagnostic test of CRC32 calculator in FPGA failed**

```
SrcNameSuffix = _SWTargets;  
Message = "(4056) Cyclic test of CRC32 calculator failed in { 1 }";  
{ 1 } = Cause of failure. Example: checkFailed, timeout  
Severity Level = Critical;
```

System Simple Event **Switch PM is performed via errorHandler**

```
SrcNameSuffix = _SWTargets;  
Message = "(4057) Failure in SM detected by PM";  
Severity Level = Critical;
```

System Simple Event **Switch PM is performed via errorHandler due to Bus Error**

```
SrcNameSuffix = _SWTargets;  
Message = "(4058) Try to switch PM due to Bus Error";  
Severity Level = Critical;
```

System Simple Event **CPU interface error in MBM1**

```
SrcNameSuffix = _SWTargets;  
Message = "(4059) CPU interface error in FPGA";  
Severity Level = Critical;
```

Events from the MMU

System Simple Event **Software errors**

```
SrcNameSuffix = _SWTargets;  
Message = "(4060) Software error detected by MMU";  
Severity Level = Fatal;
```

System Simple Event Memory violation

```
SrcNameSuffix = _SWTargets;  
Message = "(4061) Attempted write access in write-protected memory";  
Severity Level = Fatal;
```

System Simple Event MMU checker error

```
SrcNameSuffix = _SWTargets;  
Message = "(4062) Unexpected write in protected memory";  
Severity Level = Critical;
```

System Simple Event DMA checker error

```
SrcNameSuffix = _SWTargets;  
Message = "(4063) DMA Checker time. Test failed";  
Severity Level = Critical;
```

System Simple Event Primary CPU: DMA memory violation

```
SrcNameSuffix = _SWTargets;  
Message = "(4064) Primary CPU: DMA memory violation at {2}"  
{2} = General fail address information  
SeverityLevel = High
```

Events from FW Integrity Verification

Indication that FW CRC did not match original in primary PM.

```
SrcNameSuffix = _SWTargets;  
Message = "(4070) FW Integrity Verification primary: CRC did not  
match original";  
Severity Level = Medium;
```

Indication that FW CRC did not match original in backup PM.

```
SrcNameSuffix = _SWTargets;  
Message = "(4071) FW Integrity Verification backup:CRC did not match  
original";  
Severity Level = Medium;
```

Indication that FW CRC did not match in stand alone PM.

```
SrcNameSuffix = _SWTargets;  
Message = "(4072) FW Integrity Verification standalone:CRC did not  
match original";  
Severity Level = Medium;
```

Address parameter failure in FW Integrity Verification.

```
SrcNameSuffix = _SWTargets;  
Message = "(4073) FW Integrity Verification: Address parameter  
failure";  
Severity Level = Medium;
```

System Simple Event **CRC error in FW Integrity Verification**

```
SrcNameSuffix = _SWTargets;  
Message = "(4074) FW Integrity Verification trainee CRC did not match  
original"  
SeverityLevel = Critical
```

Events from the Heap: Software Errors

```
SrcNameSuffix = _SWTargets;  
Message = "(4080) Software error detected by Heap manager";  
Severity Level = Fatal;
```

Events from the Heap: Memory Violation

```
SrcNameSuffix = _SWTargets;  
  
Message = "(4081) Heap violation during allocation of an element";  
Severity Level = Fatal;
```

Message = "(4082) Heap violation during deallocation of an element";
Severity Level = Fatal;

Message = "(4083) Null element is deallocated in the heap";
Severity Level = Fatal;

Message = "(4084) Corrupt element is deallocated in the heap";
Severity Level = Fatal;

Message = "(4085) Corrupt elements are detected after a power fail";
Severity Level = Fatal;

Message = "(4086) The Protected Heap is out of memory";
Severity Level = Low;

Message = "(4087) The Shared Heap is out of memory";
Severity Level = Low;

Message = "(4093) The max boundary size of an element is exceeded in the Shared Heap";
Severity Level = Medium;

Message = "(4094) The max boundary size of an element is exceeded in the Protected Heap";
Severity Level = Medium;

Events from the Heap: Heap Checker Error

System Simple Event MemFree error - CPU Switch

SrcNameSuffix = _SWTargets;
Message = "(4088) Heap Checker detects a corrupt element during deallocation of an element";
Severity Level = Critical;

System Simple Event MemFree error - no CPU Switch

SrcNameSuffix = _SWTargets;
Message = "(4089) Heap Checker detects a corrupt element during deallocation of an element";
Severity Level = Fatal;

System Simple Event Synchronous heap check error - logging

```
SrcNameSuffix = _SWTargets;  
Message = "(4090) Corrupt element during synchronous heap check";  
Severity Level = Low;
```

System Simple Event Cyclic heap check error - CPU Switch

```
SrcNameSuffix = _SWTargets;  
Message = "(4091) Corrupt element during cyclic heap check";  
Severity Level = Critical;
```

System Simple Event Cyclic heap check error - no CPU Switch

```
SrcNameSuffix = _SWTargets;  
Message = "(4092) Corrupt element during cyclic heap check";  
Severity Level = Fatal;
```

System Simple Event Max boundary size exceeded in the Shared Heap

```
SrcNameSuffix = _SWTargets  
Message = "(4093) The max boundary size of an element is exceeded in  
the Shared Heap."  
SeverityLevel = Medium
```

System Simple Event Max boundary size exceeded in the Protected Heap

```
SrcNameSuffix = _SWTargets  
Message = "(4094) The max boundary size of an element is exceeded in the  
Protected Heap."  
SeverityLevel = Medium
```

Events from Irq Supervisor

These messages are short (twelve characters) since most of them have to be printed from interrupt context when an irq error has occurred, which means there is only a very small time margin.

```
SrcNameSuffix = _SWTargets;
```

Message = "(4100) Irq error. Unable to spawn Reset Irq Supervisor thread";

Severity Level = Medium;

Message = "(4101) Irq error. MSCallout array full; not possible to add SuperviseIrq; the IrqSupervision thread will be suspended";

Severity Level = Medium;

Message = "(4102) Irq error. Irq supervisor: Irq timed out; primary PM will be shut down";

Severity Level = Medium;

Message = "(4103) Irq error. Irq supervisor: Irq timed out; backup PM was shut down";

Severity Level = Medium;

Message = "(4104) Irq error. Irq supervisor: Irq timed out error in standalone PM";

Severity Level = Medium;

Message = "(4105) Irq error. Unable to create a OS periodic timer, the IrqSupervision thread will be suspended";

Severity Level = Medium;

Message = "(4106) Irq error. Unable to raise thread priority, the IrqSupervision thread will be suspended";

Severity Level = Medium;

Message = "(4107) Irq supervisor: Irq timed out; trainee PM was shut down"

SeverityLevel = Medium

Events from CEX Bus Interrupt Handler

SrcNameSuffix = _SWTargets;

Message = "(4110) Hanging CEX IRQ: All CEMs on the upper CEX bus segment are disabled";

Severity Level = Medium;

Message = "(4111) Hanging CEX IRQ: All CEMs on the lower CEX bus segment are disabled";

Severity Level = Medium;

Message = "(4112) Hanging CEX IRQ: The upper PM has been shut down";

Severity Level = Critical;

Message = "(4113) Hanging CEX IRQ: The lower PM has been shut down";

Severity Level = Medium;

Message = "(4115) Invalid IRQ CEM {1}: All CEMs on this CEX bus segment are disabled";

{1} = Module number of interrupting CEM

Severity Level = Medium;

Message = "(4116) Invalid IRQ CEM {1}: All CEMs on this CEX bus segment are disabled";

{1} = Module number of interrupting CEM.

Severity Level = Medium;

Message = "(4117) Invalid CEX IRQ backup PM: The upper PM has been shut down";

Severity Level = Medium;

Message = "(4118) Invalid CEX IRQ backup PM: The lower PM has been shut down";

Severity Level = Medium;

Message = "(4119) Spurious CEX IRQ: {1} spurious IRQs since system startup";

{1} = Number of spurious IRQ since start

Severity Level = Low;

Message = "(4120) Hanging CEX IRQ: All CEMs on the dir CEX bus segment are disabled";

Severity Level = Medium;

Message = "(4121) Hanging CEX IRQ: All CEMs on the indir CEX bus segment are disabled";

Severity Level = Medium;

Message = "(4122) Hanging CEX IRQ: The PM has been shut down";
Severity Level = Critical;

Message = "(4123) Invalid IRQ CEM {1}: All CEMs on this CEX bus segment are disabled";
{1} = Module number of interrupting CEM
Severity Level = Medium;

Message = "(4124) Invalid IRQ CEM {1}: All CEMs on this CEX bus segment are disabled";
{1} = Module number of interrupting CEM
Severity Level = Medium;

Message = "(4125) Insufficient memory to create the Reset BC thread";
Severity Level = Medium;

Events from DMA Supervisor

SrcNameSuffix = _SWTargets;

Message = "(4126) Error in DMA Supervisor configuration";
Severity Level = Fatal;

Events from Internal Diagnostics Engine

SrcNameSuffix = _SWTargets;

Message = "(4130) Software error detected by Diagnostic Engine";
Severity Level = Medium;

Message = "(4131) Diagnostic Engine: FDRT deadline passed";
Severity Level = Medium;

Message = "(4132) Diagnostic Engine: Diurnal deadline passed";
Severity Level = Medium;

Events from RAMTest

SrcNameSuffix = _SWTargets;

Message = "(4133) RAMTest Primary Parity error self test";
Severity Level = Critical;

Message = "(4134) RAMTest Backup Parity error self test";
Severity Level = Critical;

Message = "(4135) RAMTest Standalone Parity error self test";
Severity Level = Critical;

Message = "(4136) RAMTest Primary Address line test 0x{1}";
{1} = Fail address
Severity Level = Critical;

Message = "(4137) RAMTest Backup Address line test 0x{1}";
{1} = Fail address
Severity Level = Critical;

Message = "(4138) RAMTest Standalone Address line test 0x{1}";
Severity Level = Critical;
{1} = Fail address

Message = "(4139) RAMTest Primary Internal error";
Severity Level = Fatal;

Message = "(4140) RAMTest Backup Internal error";
Severity Level = Fatal;

Message = "(4141) RAMTest Standalone Internal error";
Severity Level = Fatal;

Events from the RCU CRC Checker

SrcNameSuffix = _SWTargets;

Message = "(4142) Hardware error detected by RCU CRC Checker";
Severity Level = Critical;

Events from RAMTest

Message = "(4143) RAMTest Trainee Parity error self test"
SeverityLevel = Critical

Message = "(4144) RAMTest Trainee Address line test 0x{1}"
{1} = Fail address
SeverityLevel = Critical

```
Message = "(4145) RAMTest Backup Internal error"  
SeverityLevel =Critical
```

Events from SSPActiveTest

```
Message = "(4146) SSP error detected by SSPActiveTest"  
SeverityLevel = Fatal
```

Events from HWSetupVerification

These events are issued if HW Setup Verification detected an error in HW Setup. The message also contains a test label, specifying the failing test.

```
SrcNameSuffix = _SWTargets;  
  
Message = "(4150) HW Setup Verification in Primary: {1}";  
{1} = Subtest strings used to specify the failing test method.  
Severity Level = Medium;  
  
Message = "(4151) HW Setup Verification in Backup: {1}";  
Severity Level = Medium;  
{1} = Subtest strings used to specify the failing test method.  
  
Message = "(4152) HW Setup Verification in Standalone: {1}";  
{1} = Subtest strings used to specify the failing test method.  
Severity Level = Medium;  
  
Message = "(4153) HW Setup Verification in Trainee: {1}"  
{1} = Subtest strings used to specify the failing test method.  
SeverityLevel = Critical
```

Events from EXTCLKSupervision

These events are issued from the EXTCLK Supervision if either the EXTCLK frequency is or the FPGA divider is working incorrect.

```
SrcNameSuffix = _SWTargets;  
  
Message = "(4160) EXTCLK Error Allowed range {1} us";  
{1} = Sleep-time information  
Severity Level = Medium;
```

```
Message = "(4161) EXTCLK Supervision Error: FATAL error";  
Severity Level = Medium;
```

Events from HRESETSupervision

This event is issued from the Oscillator Supervision task if either the SPPL or EXTCLK frequency is working incorrect.

```
SrcNameSuffix = _SWTargets;  
Message = "(4170) HRESET Error asserted by {1}";  
{1} = Strings used to specify the signals generating HRESET  
Severity Level = High;
```

Events from Modulebus Driver

System Simple Event **Comparison of CRC32 from SM and PM failed**

```
SrcNameSuffix = _SWTargets;  
Message = "(4180) MBM1 SM vs PM CRC32 failed, address 0x{1}";  
{1} = Address (hexadecimal)  
Severity Level = Low;
```

System Simple Event **Failed to create SMDrv in Modulebus**

```
SrcNameSuffix = _SWTargets;  
Message = "(4181) Failed to create SMDrv From Modulebus";  
Severity Level = Medium;
```

System Simple Event **BusErrorIn interrupt routine**

```
SrcNameSuffix = _SWTargets;  
Message = "(4182) Bus Error In Modulebus ISR address 0x{1}";  
{1} = Address (hexadecimal)  
Severity Level = Critical;
```

System Simple Event **BS Exception in MBM1scanner**

```
SrcNameSuffix = _SWTargets;  
Message = "(4183) BS EXCEPTION In MBM1 Scanner";  
Severity Level = Critical;
```

Message = "(4184) Incoming safety header failure, address 0x{1}"
{1} = Address (hexadecimal)
SeverityLevel = Medium

Message = "(4185) Primary shutdown due to suspect SM"
SeverityLevel = Medium

Message = "(4186) No answer from SM address 0x{1}, error code 0x{2}"
SeverityLevel = Medium

Message = "(4187) Failure in safety IO, address 0x{1}, error code 0x{2}"
{1} = Address (hexadecimal)
{2} = Error code (hexadecimal)
SeverityLevel = Medium

Events from ModuleBus

System Simple Event

Message = "(4901) Event overflow in module: {1}{2}"
{1} = Path to ModuleBus unit.
{2} = Unit number.
SeverityLevel = Medium

Controller – Hardware

Hardware generated system alarms are automatically available when the hardware is configured. They may however be disabled.

All Hardware Units in the hardware configuration have one system alarm and one system simple event each for its disposal. The intention is to have a sum alarm and a sum event for different errors and warnings that can be detected on the hardware unit.

Table 62. Parameters for Hardware Generated System Alarms and Events

Parameters	Descriptions
Class	All hardware generated system alarms and events have the same value of parameter 'Class' that is determined by the value of CPU setting 'AE System AE class'.
Severity	Values of severity are defined through the CPU setting 'AE System AE high severity' for hardware generated system alarms, respective 'AE System AE medium severity' for hardware generated system simple events.

Table 62. Parameters for Hardware Generated System Alarms and Events

Parameters	Descriptions
Message	<p>The message contains reference to more detailed information, because each alarm is a sum alarm that can indicate many different errors on the unit. This information is given in the description of Errors and Warnings in Control Builder and/or in the System status viewer in Plant Explorer.</p> <p>The error code is stored in two 32 bit words first word is <i>ErrorsAndWarnings</i> and the second is <i>ExtendedStatus</i>.</p> <p>In each hardware generated system alarm or event message, <i>ErrorsAndWarnings</i> and <i>ExtendedStatus</i> bit patterns are translated into a text in the OPC-server. General status bits are translated into a explaining text e.g. "I/O configuration error". Device specific bits from <i>ErrorsAndWarnings</i> are translated into a text in the OPC-server, if a matching text is available in the hardware definition. Otherwise they are displayed as "Device specific bit xx" in the message e.g. "Device specific bit 31". The same goes for <i>ExtendedStatus</i>. If a matching text is not available in the hardware definition, unit specific bits from <i>ExtendedStatus</i> are displayed as "Extended status bit xx" in the message e.g. "Extended status bit 0". The bits for every unit are explained later in this section.</p> <p>Example "Controller_1 (0000) I/O configuration error, Device specific bit 31, Extended status bit 0"</p> <p>If the Unit in this example is a PM865, "Device specific bit 31" = "Battery low" and "Extended status bit 0" = "Backup CPU stopped"</p> <p>In the controller log <i>ErrorsAndWarnings</i> and <i>ExtendedStatus</i> are presented as HEX format.</p> <p>Example: "E 2004-03-08 10:25:06.677 On Unit= 2 HWError Controller_1 Errorcode=16#80004000 16#00000001 (0000) See HW-tree"</p>

Table 62. Parameters for Hardware Generated System Alarms and Events

Parameters	Descriptions
SrcName	The syntax for the source name in the SrcName parameter is dynamically based on the IP address together with the SrcNameSuffix that is the hardware unit address in the hardware tree configuration. Example: IP address (172.16.85.33) + SrcNameSuffix (2.5.101) = "172.16.85.33-2.5.101".
CondName	All hardware generated system alarms have "HWEError" as common condition name in the CondName parameter.
AckRule	Ack Rule 5 is used for these system alarms,.

Alarms and Events Common for all Units

Table 63 lists those status bits that have the same meaning for all hardware units.

Note however that different units have different capabilities. A specific unit will typically only be able to generate alarms and events for an assortment of the common status bits.

Table 63. General status bit ErrorsAndWarnings

Bit	Status Bit	Value	Indication	Generation	Severity	Description
0	ConnectionDown	16#00000001	Error	Alarm	High	Connection down
1	IoError	16#00000002	Error	Alarm	Medium	I/O error
2	ModuleMissing	16#00000004	Error	Alarm	High	Module missing
3	WrongModuleType	16#00000008	Error	Alarm	High	Wrong module type
4	StatusChannelError	16#00000010	Warning	Alarm	Medium	Channel error
5	IoWarning	16#00000020	Warning	Event	Low	I/O warning
6	StatusUnderflow	16#00000040	Warning	Alarm	Low	Underflow
7	StatusOverflow	16#00000080	Warning	Alarm	Low	Overflow
8	StatusForced	16#00000100	Warning	Event	Low	Forced

Table 63. General status bit ErrorsAndWarnings (Continued)

Bit	Status Bit	Value	Indication	Generation	Severity	Description
9	WatchdogTimeout	16#00000200	Error	Alarm	High	Watchdog timeout
10	DeviceFailure	16#00000400	Error	Alarm	High	Device failure
11	DeviceNotFound	16#00000800	Error	Alarm	High	Device not found
12	WrongDeviceType	16#00001000	Error	Alarm	High	Wrong device type
13	IOConnectError	16#00002000	Error	Alarm	Medium	I/O connection error
14	IOConfigError	16#00004000	Error	Alarm	Medium	I/O configuration error
15	HWConfigError	16#00008000	Error	Alarm	High	Hardware configuration error
16	GeneralError ¹	16#00010000	Error	–	–	–
17	GeneralWarning ¹	16#00020000	Warning	–	–	–
18	RedWarningPrimary ²	16#00040000	Warning	Event	Low	Warning on primary unit
19	RedWarningBackup ²	16#00080000	Warning	Event	Low	Warning on backup unit
20	RedErrorBackup ²	16#00100000	Warning	Alarm	Medium	Error on backup unit
21	<i>Reserved</i>	16#00200000	–	–	–	–
22	DeviceSpecific10	16#00400000	3	3	3	3
23	DeviceSpecific9	16#00800000	3	3	3	3
24	DeviceSpecific8	16#01000000	3	3	3	3
25	DeviceSpecific7	16#02000000	3	3	3	3
26	DeviceSpecific6	16#04000000	3	3	3	3
27	DeviceSpecific5	16#08000000	3	3	3	3
28	DeviceSpecific4	16#10000000	3	3	3	3

Table 63. General status bit ErrorsAndWarnings (Continued)

Bit	Status Bit	Value	Indication	Generation	Severity	Description
29	DeviceSpecific3	16#20000000	3	3	3	3
30	DeviceSpecific2	16#40000000	3	3	3	3
31	DeviceSpecific1	16#80000000	3	3	3	3

- 1 Used together with other status bits.
- 2 Used only if hte unit is configured as a redundant unit.
- 3 Depends on the specific hardware device ,defined within hardware definition file.

Table 64. General status bit ExtendedStatus

Bit	Status Bit	Value	Indication	Generation	Severity	Description
0	ExtendedStatus1	16#00000001	1	1	1	1
1	ExtendedStatus2	16#00000002	1	1	1	1
2	ExtendedStatus3	16#00000004	1	1	1	1
3	ExtendedStatus4	16#00000008	1	1	1	1
4	ExtendedStatus5	16#00000010	1	1	1	1
5	ExtendedStatus6	16#00000020	1	1	1	1
6	ExtendedStatus7	16#00000040	1	1	1	1
7	ExtendedStatus8	16#00000080	1	1	1	1
8	ExtendedStatus9	16#00000100	1	1	1	1
9	ExtendedStatus10	16#00000200	1	1	1	1
10	ExtendedStatus11	16#00000400	1	1	1	1
11	ExtendedStatus12	16#00000800	1	1	1	1
12	ExtendedStatus13	16#00001000	1	1	1	1
13	ExtendedStatus14	16#00002000	1	1	1	1
14	ExtendedStatus15	16#00004000	1	1	1	1
15	ExtendedStatus16	16#00008000	1	1	1	1
16	ExtendedStatus17	16#00010000	1	1	1	1

Table 64. General status bit ExtendedStatus (Continued)

Bit	Status Bit	Value	Indication	Generation	Severity	Description
17	ExtendedStatus18	16#00020000	1	1	1	1
18	ExtendedStatus19	16#00040000	1	1	1	1
19	ExtendedStatus20	16#00080000	1	1	1	1
20	ExtendedStatus21	16#00100000	1	1	1	1
21	ExtendedStatus22	16#00200000	1	1	1	1
22	eA	16#00400000	–	–	–	–
23	PrimaryIncompatibleFW ²	16#00800000	Error	Alarm	High	Version of the Running Primary is incompatible
24	BackupIncompatibleFW ²	16#01000000	Warning	Alarm	Medium	Version of the Running Backup is incompatible
25	PrimaryNotPrefrdFW ²	16#02000000	Warning	Alarm	Medium	Version of the Running Primary is not preferred
26	BackupNotPrefrdFW ²	16#04000000	Warning	Alarm	Medium	Version of the Running Backup is not preferred
27	TimeouOnBackup ²	16#08000000	Warning	Alarm	Low	Watchdog timeout on backup
28	DeviceFailureBackup ²	16#10000000	Warning	Alarm	Low	Backup device failure

Table 64. General status bit ExtendedStatus (Continued)

Bit	Status Bit	Value	Indication	Generation	Severity	Description
29	SwitchoverInProgress ²	16#20000000	Warning	Event	Low	Switchover in progress
30	ConfiguredAsRedundant ^{2, 3}	16#40000000	-	-	-	Redundant mode enabled
31	UnitBPrimary ²	16#80000000	-	-	-	Unit B acts primary

1 Depends on the specific hardware device, defined within the hardware definition file.

2 Used only if the unit is configured as a redundant unit.

3 If this bit is set and bit 31 is not set, the text *Unit A acts primary* will be shown.

Unit Specific Alarm and Events

The status bit description of alarms and events specific to a hardware unit is available along with the hardware type displayed under the hardware library.

To view the status bit description of alarms and events specific to a hardware unit:

1. Go to **Libraries > Hardware > HwLib** folder, and expand the **Hardware Types** folder.
2. Right click the required hardware type and select **Unit Specific Alarm and Events** (see [Figure 218](#)).

The status bit description of alarms and events is displayed in a separate window (see [Figure 219](#)). This data can be used for supervising the unit status, see [Supervising Unit Status](#) on page 375.

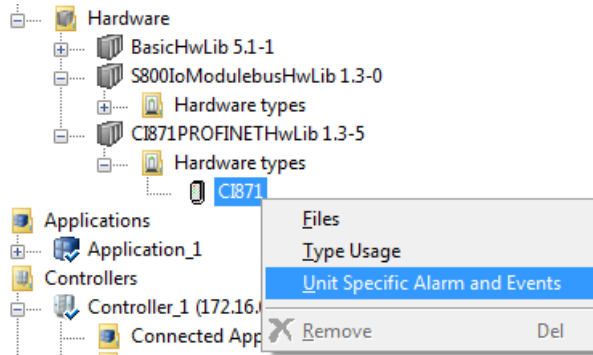


Figure 218. Accessing the Unit Specific Alarm and Events option

Unit Specific Alarm and Events - CI871

Bit	Status Type	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Event	High	Communication problems due to flooding
27	ErrorsAndWarnings	Error	Event	High	PNIO Alarms blocked
28	ErrorsAndWarnings	Error	Event	High	CEX watchdog
29	ErrorsAndWarnings	Warning	Event	High	Communication memory obtained too lo
30	ErrorsAndWarnings	Error	Alarm	High	Ethernet cable dropped
31	ErrorsAndWarnings	Error	Event	High	Hardware failure

Figure 219. Unit Specific Alarm and Events window for CI871 hardware type

A

- abort
 - tasks 183
- access control
 - double authenticate 133
- access variables 97
- accuracy 288
- acknowledge
 - errors 387
 - warnings 387
- acknowledgement rules 264, 317
- add
 - files to FSD server 402
 - message to alarm 264
 - to libraries 147
- AF 100 357
- afw file 415
- alarm condition
 - name 264
- alarm conditions 260
- Alarm handling
 - INSUM 281
- Alarm lists
 - INSUM 281
- AlarmCond 260
 - parameters 261
- AlarmCondBasic 262
- AlarmCondBasicM 262
- AlarmCondM 260
- alarms
 - add message 264
 - classify 264
 - communication 289
 - condition name 264
 - disable 270
 - disable condition 264
 - enable 263
 - examples 297
 - in control loops 270
 - inhibit 270
 - local printer 293
 - logging 285
 - severity 264
 - source name 264
 - state diagrams 317
 - status 266
 - subscribe to 289
 - system 282
- all inputs 378
- all outputs 378
- all unit status 377
- analog signals
 - scale 375
- analysis tools 477
- applicable specification 18
- applications
 - export 415
 - rollback 415
- arrays
 - example 495
- ASCII codes 512
- ASCII conversion 514
- aspect object
 - set 80
 - set attribute 80
 - suppress 81
- aspect objects 80
- aspects
 - Library Version Definition 414
- asynchronous communication 354

attributes
 aspect object 80
 Hidden 158
 initial value 96
 override 134
 Protected 158
authentication
 at download 196
 levels 134
authentication at download
 enable 196

B

backup 401
 individual files 402
BasicLib 42
buffer queues 291
buffers
 configure 291
 memory planning 292

C

CASE 234
change
 library state 143
 library version 145
Change Library Version 145
channel status
 check 378
check
 channel status 378
classify
 alarms 264
client/server 352
Closed 142
CNCP 288
codes
 ASCII 512
COMLI 324
communication

 alarm and event 289
 client/server 352
 cyclic vs. asynchronous 354
 function blocks 350
 libraries 324, 350
 master/slave 352
 modem 334
 provider/subscriber 353
 serial 336
 SIL restrictions 323
 statistics 343
 using access variables 99
 using global variables 100
 variables 343
communication libraries
 library 328
Communication Variables 101, 343
 Diagnostics 471
complex types
 modify 58
condition name
 alarms 264
condition-related events 258
configure
 buffers 291
 Error Handler 419
 OPC AE communication 289
Confirmed Online Write 195
connect
 I/O channels 122
 libraries 136
 library 138
 objects 66
 to other system 348
Control Builder
 start log 454
 system information report 457
Control Builder start log
 file path 454
control loops

- alarms 270
- control module types
 - AlarmCondBasicM 262
 - AlarmCondM 260
- control modules
 - execution 73
 - single 76
- Control Network 341
 - network areas 341
- control project
 - create 22
 - insert library 138
 - remove library 140
- control projects
 - projects 22
- controller logs
 - file paths 462
- controller system log 458
- controllers
 - export 415
 - rollback 415
 - system alarms 283
- conversion
 - ASCII 514
- conversion functions
 - example 510
- conversions
 - ASCII 512
- Coordinated Universal Time 287
- crash dumps 467
- create
 - control project 22
 - library 141
 - new library version 145
 - objects 66
- Create New Library Version 145
- cyclic communication 354
- read 348
- send 348
- data types 86
 - simple 86
- debug mode 186
- decisions
 - when creating types 64
- declare
 - external variables 47
 - function blocks 48
 - parameter 46
 - types 45
 - variables 46
- define
 - variables 85
- delete
 - library 141
- Deploy 27, 29
- development state
 - libraries 142
- Device Import Wizard 152
- DeviceNet EDS 152
- Diagram and Diagram Types
 - Execution 74
 - Introduction 59
- dialogs
 - Remote System 342
- disable
 - alarm condition 264
 - alarms 270
 - authentication at download 196
 - events 270
- disconnect
 - library 140
- document conventions 16
- Double Authenticate 133
- download
 - reports 248

D

data

E

- EDIT parameters 262
- editors
 - declare types 45
 - diagram 52
 - graphics 53
 - programming 50
- enable
 - alarm detection 263
 - authentication at download 196
- Engineering Environment 27
- enter
 - variables 85
- Entity 24
 - Deploy 29
- Environment
 - Engineering 27
 - Production 27
- Environments 27
- Error Handler
 - configure 419
 - log entries 430
- error messages 390
- Error Reaction
 - settings 425
- error reports 493
- errors
 - acknowledge 387
 - non-unique alarms 265
 - suggested actions 485
 - symptoms 485
- Ethernet/IP EDS 152
- events
 - communication 289
 - condition-related 258
 - disable 270
 - inhibit 270
 - logging 285
 - low level 274
 - simple 258
 - system 282
 - tracking-related 258
- examples
 - alarm and event 297
 - arrays 495
 - conversion functions 510
 - queues 506
 - type protection 161
- execution
 - control for individual objects 113
 - control modules 73
 - control using external variables 113
 - control using parameter 114
 - debug mode 186
 - non-cyclic 186
- EXIT 234
- export 413
 - alternatives 416
 - application 415
 - applications 415
 - controller 415
 - controllers 415
 - libraries 413, 418
 - library 141, 413
- extensible parameters 127
- external 96
- external time stamps 286
- external variables 96
 - declare 47

F

- Fatal Overrun 428
- fatal overrun 425
- fault localization
 - crash dumps 467
- FDRT 428
- file paths
 - Control Builder start log 454
 - controller logs 462
 - heap statistics log 458

- session logs 451
- firmware functions 30
- folders
 - System 30
- FOR 234
- force
 - I/O channels 198
- FOUNDATION Fieldbus
 - HSE 324
- Foundation FIELDBUS
 - HSE 324
- FSD server
 - add files 402
 - remove files 402
- FSD util tool 402
- function block types
 - AlarmCond 260
 - AlarmCondBasic 262
 - System Diagnostics 479
 - SystemDiagnostics 316
- function blocks
 - communication 350
 - declare 48
- functions 30

G

- graphical editor 53

H

- hardware
 - monitor 126
- hardware status 376
- heap 482
- heap statistics log 458
 - file path 458
- Heap Utilization 483
- Hidden
 - attribute 158

I

- I/O addressing 121
- I/O channels
 - connect 122
 - force 198
 - force values 372
 - monitor 126
 - online mode 372
- IAC MMS 344
- IEC 61131-3 23
- Implicit Cast 234
- import 413
 - alternatives 416
 - library 141
- inhibit
 - alarms 270
 - events 270
- init values
 - set 95
- Init_Val 95
- initial values 94
 - priority order 95
 - retain attributes 96
- Instruction List 234
- INSUM 326
 - Alarm handling method 281
 - Alarm lists 281
- INSUM Alarms 276
- InsumCommLib 328
- interaction windows 388
- internal time stamps 286
- interval time
 - tasks 172
- intervals
 - tasks 172

K

- keywords
 - in parameter descriptions 128

L

Ladder Diagram 234
latency 178, 181
latency supervision 424
levels
 authentication 134
libraries
 add to 147
 Alarm and Event 259
 BasicLib 42
 communication 324, 350
 connect 136
 connect to application 138
 connect to library 138
 create 141
 delete 141
 disconnect 140
 export 141, 413, 418
 import 141
 insert into control project 138
 InsumCommLib 328
 management 135
 remove from control project 140
 states 142
 version handling 143, 146
library management 135
library state
 change 143
library states
 Closed 142
 Open 142
 Released 142
library version
 change 145
 create new 145
load balancing 184, 424
Load Firmware 226
local printers 293
log
 alarms and events 285

 simple events 286
log entries
 Error Handler 430
log files 447
 Control Builder start log 454
 controller system log 458
 heap statistics 458
 session 448
 system log 448
logging
 alarms and events 285
Loops In ST 234
low level event 274

M

maintenance
 tools 418
master
 time synchronization 288
master/slave 352
maximum number of forced I/O channels 374
MB 300 TS 288
MB300 332
memory size 482
MMS 333
MMS Time Service 288
ModBus 334
modem communication 334
monitor
 hardware 126
 I/O channels 126
MTMCommLib 335
Multi-User Engineering 24

N

National Language Support 296
Nested If or Case 234
network areas 341
network redundancy 341
NLS

- syntax 296
- non-cyclic execution 186
- non-unique alarms
 - errors 265
- number conversion
 - example 510

O

- objects 43
 - connect 66
 - control execution 113
 - create 66
- offset
 - tasks 173
- online mode 126
 - all inputs/outputs 378
 - all unit status 377
 - force I/O channels 372
 - hardware status 376
 - I/O channels 372
 - interaction windows 388
 - messages 390
 - project documentation 395
 - search and navigation 391
 - status indications 386
 - task overview 387
 - unit status 375
- OPC AE communication
 - configure 289
- OPC Server
 - session log example 453
- OPC server
 - subscriptions 289
- OPC Server for AC 800M 289
- Open 142
- open
 - code block menu 52
- override
 - attributes 134
 - property attributes 134

- protection 160
 - type protection 159
- overrun 178 to 179
- overrun supervision 424

P

- parameter
 - declare 46
- parameters 81
 - AckCond 263
 - AckRule 263
 - AELimit auto disable 267
 - AlarmCond 261
 - Class 264
 - CondName 264
 - CondState 266
 - DisCond 264
 - EDIT 262
 - EnCond 264
 - EnDetection 263
 - Error 266
 - extensible 127
 - ExtTimeStamp 263
 - FilterTime 263
 - Inhibit 272
 - Inverted 263
 - keywords 128
 - Message 264
 - Severity 264
 - Signal 263
 - SignalID 263
 - SrcName 264
 - Status 266
 - TransitionTime 286
 - UseSigToInit 263
- permissions
 - Re-Authenticate 133
- POU 23
 - definition 23
- printers

- local 293
- priorities
 - tasks 169
- priority
 - initial values 95
- priority order
 - initial values 95
- process alarms 259
- Production Environment 27
- PROFIBUS
 - DP-V1 355
- PROFIBUS GSD 152
- PROFINET IO 356
 - C871 356
 - GSD 356
- PROFINET IO GSD 152
- programming editor 50
- project
 - insert library 138
 - remove library 140
- project constants 116
 - structured 117
- project documentation 253, 395
- property attributes
 - override 134
- property permissions 133
 - set 134
- Protected
 - attribute 158
- protection
 - example 161
 - override 160
 - override for types 159
- Protocols
 - PROFINET IO 356
- protocols
 - COMLI 324
 - FF HSE 324
 - INSUM 326
 - MB300 332

- MMS 333
- ModBus 334
- modem communication 334
- SattBus 335
- serial communication 336
- Siemens S3964R 334
 - supported 340
- provider/subscriber 353
- publisher/subscriber 353

Q

- queues
 - buffer 291
 - example 506

R

- Reaction
 - settings for HI controller 427
- read
 - data 348
- Re-Authenticate 133
- Re-authenticate 133
- redundancy
 - network 341
- Released 142
- Remote System dialog 342
- remote systems information 468
- remove
 - files from FSD server 402
- REPEAT 234
- reports
 - at download 248
 - system information 457
- Reservation 24
 - Release 25
 - Reserve Entity 25
- resolution 288
- restore 401
- retain attributes
 - initial values 96

Reverse attribute 108
rollback
 application 415
 controller 415
RS-232C 341
RT 356

S

S3964R 334
SattBus 335
scale
 analog signals 375
search and navigation 198
 online mode 391
Secure Digital 218
Security 195
security
 double authenticate 133
 re-authenticate 133
self-defined types 159
send
 data 348
Sequence-of-Events (SOE) 274
serial communication 336
session log
 OPC server example 453
session log files 448
session logs
 file paths 451
set
 aspect object 80
 aspect object attribute 80
 authentication level 134
 property permissions 134
 specific initial values 95
settings
 import/export 416
severity
 alarms 264
SFC 234

Siemens
 S3964R protocol 334
SIL
 communication 323
 Confirmed Online Write 195
 no disabling of alarms via MSS 271
simple data types 86
simple events 258
 log 286
Simultaneous Execution in SFC 234
single control modules 76
SNTP 288
Source Code Report 245
source name
 alarms 264
specific initial values
 set 95
start code blocks 52
state
 libraries 142
state diagrams
 alarms 317
 alarms,alarms
 state diagrams 317
statistics
 communication 343
status
 alarms 266
 indications 386
status messages 390
structured project constants 117
subscribe
 to alarms 289
sum system alarms 320
supervise
 hardware 126
 I/O channels 126
 unit status 375
supervision
 latency 181

- overrun 179
- supported protocols 340
- suppress
 - aspect object 81
- syntax
 - NLS 296
- System alarms
 - List 283
- system alarms
 - controller generated 283
 - sum 320
- system alarms and events 282
- System Diagnostics 342, 479
- system diagnostics 316, 479
- System folder 30
- system information 457
- system log file 448

T

- Task Analysis 187
- tasks 387
 - abort 183
 - execution 169
 - interval time 172
 - offset 173
 - priorities 169
 - time-critical 169
- TCPCommLib 338
- terminology 18
- time stamps 286
 - external 286
 - internal 286
- time synchronization 288
- time-critical tasks 169
- tools
 - analysis 477
 - FSD util 402
 - maintenance 418
- tracking-related events 258
- TransitionTime 286

- trouble-shooting 446
 - symptoms and measures 485
- type concept 43
- types 43
 - document 253
 - in applications 56
 - in libraries 57
 - self-defined 159

U

- UDPCCommLib 338
- UNICODE 296
- unit status
 - supervise 375
- UTC 287

V

- variable communication 343
- variables 81, 96
 - access 97
 - attributes 88
 - declare 46
 - define 85
 - initial values 94
 - list 84
- version handling
 - libraries 143, 146

W

- warnings
 - acknowledge 387
- WHILE 234

Revision History

Introduction

This section provides information on the revision history of this User Manual.



The revision index of this User Manual is not related to the 800xA 6.0 System Revision.

Revision History

The following table lists the revision history of this User Manual.

Revision Index	Description	Date
-	First version published for 800xA 6.0	August 2014
A	Version published for 6.0.2	April 2016

Updates in Revision Index A

The following table shows the updates made in this User Manual for Revision Index A.

Updated Section/Sub-section	Description of Update
Section 1, Basic Functions and Components	Added Difference Report Viewer. Added Communication Variable Limits Dialog.
Section 5, Maintenance and Trouble-Shooting	Updated Controller Configuration topic. Updated Online Upgrade topic

Contact us

www.abb.com/800xA
www.abb.com/controlsystems

Copyright © 2016 ABB.
All rights reserved.

3BSE035980-600 A

Power and productivity
for a better world™

